MEASUREMENT SYSTEM DEVELOPMENT AND ASSESSMENT OF WHOLE BODY VIBRATION TRANSMISSION IN POWER WHEELCHAIRS

Undergraduate Honors Thesis

Presented in Partial Fulfillment of the Requirements for
Graduation with Honors Research Distinction in the
Department of Mechanical Engineering at
The Ohio State University

by

L'Nard Evan Travis Tufts II

Advisors:
Sandra A. Metzler, D.Sc., P.E.
Carmen P. DiGiovine, Ph.D.

April 2015

**1. DIVISION OF OCCUPATIONAL THERAPY, 2. ASSISTIVE TECHNOLOGY CENTER,
3. DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING**

Abstract

Prolonged exposure to vibration is known to be detrimental to human health, causing pain and discomfort along the neck and spine and worsening injuries in those regions. Persons who use power wheelchairs are particularly susceptible to these dangers, many of whom have preexisting spinal conditions and as such are required to spend extended periods in a seated position. Previous studies have evaluated the effects of vibration, shock, and motion on humans in motor vehicle, industrial equipment, and even manual wheelchair applications. Little is known, however, about the particular characteristics of personal, power mobility solutions and the effects they have on users. The purpose of this study is two-fold: 1) to develop an adaptable system to enable the simultaneous collection of acceleration data from multiple points of interest (POIs) on power wheelchairs, and 2) to assess the dynamic characteristics of the power wheelchair to determine the transmissibility of whole body vibration at these points of interest. The first goal was met with the development of a nodal sensor system using the Arduino micro controller platform. The measurement system consists of a nodal unit which integrates an accelerometer with an Arduino Uno, a SD data logger to collect and temporarily store the data, and a wireless XBee RF radio to optionally stream data to a base computer. The system provides a simple, modular solution for collecting motion data from multiple locations and can be utilized for future mobility studies. To model it's dynamic characteristics, an anthropometric test device was used while the chair traversed a standardized road course. Acceleration data was collected at the frame, seat pan, and back support to develop the model. The results showed that the characteristics of the power wheelchair were unique in that the back plate is a critical location particularly for mid-wheel drive power wheelchairs. Vibration magnitudes were amplified by at

least 1.5% from the chassis frame to the the back plate and were dissipated by 34.7% from the chassis frame to the seat pan when using a 100kg wheelchair test device. While this provides some indication on the dynamic characteristics of power wheelchairs, future work will be conducted comparing multiple wheelchair models with human users.

Acknowledgements

I would like to thank several influential people and entities that contributed to the success of this project:

Dr. Sandra Metzler

Dr. Carmen DiGiovine

Kevin Wolfe

Joe West

John Bolte

Kyle Icke

Invacare Corporation

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

The term *whole-body vibration* (WBV) is defined simply as vibration affecting the entire body. “It is a term used to describe human exposure to the effects of mechanical forces such as shocks, jolts, lateral sway and vertical bouncing that are transmitted to the body”[1]. While it can affect a person in any position, such as standing or lying down, one typically experiences WBV while seated where it is transmitted through the buttocks, back, and feet. Due to these locations, the effects of prolonged exposure to WBV have been studied extensively in the occupational hazards and transportation fields where the operation of industrial equipment, motor vehicles, and public transportation produce significant levels of vibration to users and occupants.

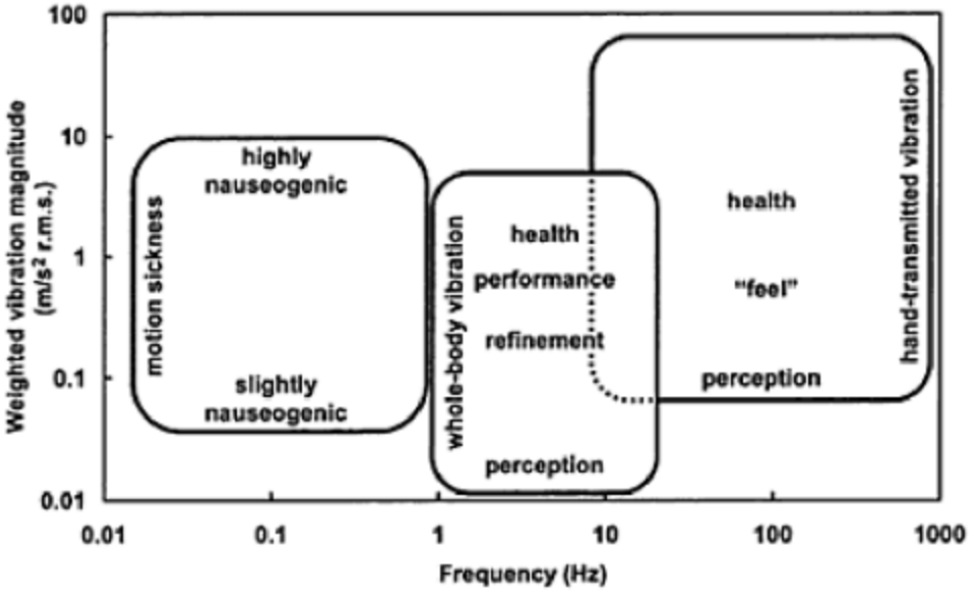


FIGURE 1
Typical frequency ranges and magnitudes of interest for the study of motion sickness, whole body vibration, and hand-transmitted vibration [2]

Humans are most sensitive to WBV within the frequency range of 1 to 20 Hz, as shown in Figure 1. Frequencies from 0.5 to 80 Hz are typically evaluated when performing WBV

analyses. Its effects are divided into one of three categories (comfort, performance, and health). The U.S. Occupational Safety and Health Administration (OSHA) recognizes WBV as a type of cumulative trauma which, based on aforementioned research in the occupational hazards and transportation fields, results in various musculoskeletal disorders (MSD). These disorders include herniated disks, degenerative disk disease, degenerative joint disease, arthritis, and severe chronic back and/or neck pain [1] in addition to varying levels of discomfort. In fact, one of the most common complaints from individuals exposed to high levels of WBV is pain along the spine [3].

Complaints of back and neck pain are echoed by users of manual wheelchairs as well. According to a 2003 study by Boninger et. al., 66% of subjects reported experiencing neck pain since becoming a person who uses a wheelchair[4]. Over the past decade and a half, there have been numerous studies and publications that investigate whether WBV was the cause of this pain for users of manual wheelchairs as well as what could be done to reduce it's negative effects. Much of this work was examined and summarized by Pearlman [3] who found:

1. WBV exposure levels experienced by users of wheelchairs exceed limits that are considered both comfortable and healthy [3]
2. Seating, suspension, and surface features each play a factor in modulating WBV levels [3]
3. WBV dose reduction to safe levels has not been achieved through current design efforts [3]

One study of note by Wolfe et. al. [5] tested suspension systems across several models of power wheelchairs (PWC) by varying suspension stiffness to assess how well each performed at

reducing WBV vibration at the seat plate surface. The study provides a look into how vibration levels vary over different surfaces and concluded that suspension systems on PWCs attenuate WBV at the seat but do not reduce it to safe levels. It also found that a mass-spring-damper model of the PWC underestimated the levels of WBV measured at the seat. Wolfe suggests that additional testing be conducted on PWCs and alternative suspension system designs.

Invacare Corporation has recently developed a prototype suspension systems for its TDX line of PWCs. They believe that this prototype system will reduce the level of WBV a user experiences and provide greater comfort for the user. Invacare has reach out to the Assistive Technology Center at The Ohio State University to research the what effect WBV has on persons who use a PWC and devise a method to test and validate their new system against existing models. The research discussed in this paper is part of a larger three phase study being conducted for the above purpose.

1.1 Focus of Thesis

The purpose of this study is: 1) to develop an adaptable system to enable the simultaneous collection of acceleration data from multiple points of interest (POIs) on power wheelchairs, and 2) to assess the dynamic characteristics of the power wheelchair to determine the transmissibility of whole body vibration at these points of interest. Using experiential knowledge with the Arduino micro-controller platform, a nodal sensor system was developed. The measurement system consists of a nodal unit which integrates an analog accelerometer with an Arduino Uno, a SD data logger to collect and store the data, and a wireless XBee RF radio to optionally stream data to a base computer. The system provides a simple, modular solution for

collecting motion data from multiple locations of a PWC and can be utilized for future mobility studies. Vibration Dose Values and Root Mean Square Accelerations were calculated to analyze the system according to ISO 2631-1: Mechanical vibration and shock - Evaluation of human exposure to whole-body vibration. From those values, the seat effective amplitude transmissibility (SEAT) was determined based on methodology from Mansfield [2].

1.2 Significance of Research

In 2011, there were 3.3 million people living in the United States who used a wheelchair over the age of 15 [6]. On average, users of power wheelchairs spend roughly 10 hours per day in their chairs [7] which according to previous research provides a significant exposure time to potentially uncomfortable and unhealthy levels of WBV. To date, there is a lack of knowledge and understanding of how and to what extent WBV is a factor on the health and comfort of PWC users. Current research provides only an analysis of the seat surface for seat cushion and suspension system transmissibility. This study provides an indication of relative magnitudes of WBV experienced at the seat pan and back plate of a PWC to capture a more full system understanding. The study also provides a method for measuring and collecting vibration data from PWCs to be used in future power mobility studies.

1.3 Overview of Thesis

This thesis will present my research work on developing a measurement system for and assessing WBV in PWC. The thesis is structured into five chapters. The fist chapter introduces the research and provides details on the focus and significance of the research. Chapter 2 details

the development of the measurement system used for this study. Chapter 3 discusses the methods used for data collection and processing. Chapter 4 offers some discussion on the results and chapter 5 concludes this thesis. Appendices can be found following the conclusion.

Chapter 2: Measurement System Development

2.1 Introduction

The objective for this phase of the study was to develop an adaptable system that would enable the simultaneous collection of acceleration data from multiple points of interest (POIs) on PWCs. The desired parameters and design constraints to be met were the following:

1. **Adaptability**: Location and number of sensors were not decided upon initially; therefore, the system must be mountable to any point(s) on the PWC. The system also has to work with any PWC since multiple PWC models will be tested as part of the overall study.
2. **Low Cost**: This system should be economically accessible for assistive technology centers and clinics which could benefit from its use. No strict budget was given, however, any choices made had to balance performance and cost.
3. **Tether Free**: It is possible to collect data using a number of sensors tethered to a base computer running data acquisition (DAQ) software such as LabVIEW. It is preferable that this system be controlled, powered, and able to collect/send data without a physical connection to any external object. The measurement device(s) should be completely integrated on the PWC.

4. **Ease of Use**: The system and process to use the system should be easy to teach and understand so that no specialized knowledge is required beyond a set of simple instructions.
5. **Modular**: The system should allow a researcher to increase or decrease (within reason) the number of POIs to be measured.

2.2 Controller Selection

When this research began, the existing instrumentation used in the Assistive Technology (AT) Center was a National Instruments NI 9234 4 channel DAQ module with cDAQ 9191 wireless chassis. It used a PCB ICP triaxial seat pad accelerometer and was able to transfer data over Wi-Fi to a computer running LabVIEW. Data was collected as long as the cDAQ was connected to a computer via ethernet cable, however, attempts to configure the system to operate over Wifi proved unsuccessful. The reason for this was due to the security configuration of the OSU Medical Center network. Furthermore, the seat pad accelerometer, NI hardware, and LabVIEW were all relatively high end measurement and data collection tools and as such were collectively quite expensive. The DAQ module and wireless chassis alone totaled over $2200 and only served one sensor location. This failed to meet the stated design parameter of low cost but, due to its specific usefulness for measuring acceleration at the interface between a PWC user and the seat cushion, development and troubleshooting was continued by another member of the research team. Meanwhile, evaluation of alternative devices commenced.

John Clark, a former student researcher on this project, had completed previous research and development work using the Arduino platform. Its website describes Arduino as “an open

source electronics platform based on easy-to-use hardware and software" [8]. Arduino micro-controllers are widely used in hobby electronics projects, but are also useful for a variety of engineering applications. Arduino boards come in a variety of models with each coming in different sizes, processing abilities, and having different number of input/output (I/O) pins. It was decided to continue to use the Arduino platform for this project due to past experience and relative proficiency using it for control projects both technical, hobby, and artistic in nature. A familiarity with its capabilities and limitations existed and there was a wealth of support, knowledge, and resources through the research advisors, staff in Ohio State's Mechanical Engineering department, and the large open source community.

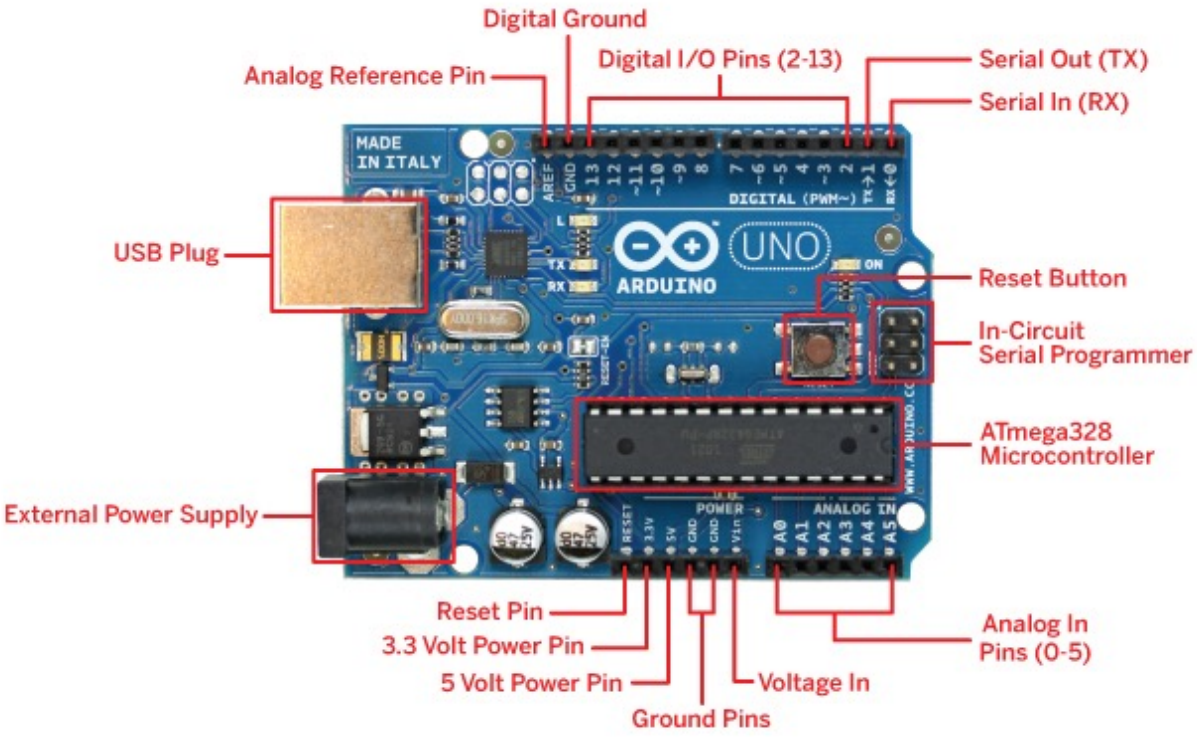


FIGURE 2
Arduino Uno R3 pinout diagram (Source: http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf)

The most widely used model of Arduino is the Arduino Uno R3, shown in Figure 2. A full pinout diagram for the Uno R3 can be found in Appendix A. This model in particular has a vast number of shields (modular circuit boards that stack onto an Arduino [9]) available to use with it which allows for the integration of several additional functions and capabilities; namely a method to wirelessly transmit data through some protocol to a remote computer. The Uno is also a relatively low cost interface ($24.95 from sparkfun.com at the time of this writing) for controlling sensors and DAQ processes compared to other offerings. Based on personal knowledge, availability of support at the university, the expansive open source community online, low cost, and availability and modularity of shields, the Arduino Uno R3 was chosen as the primary controller for the system.

2.3 Component Selection

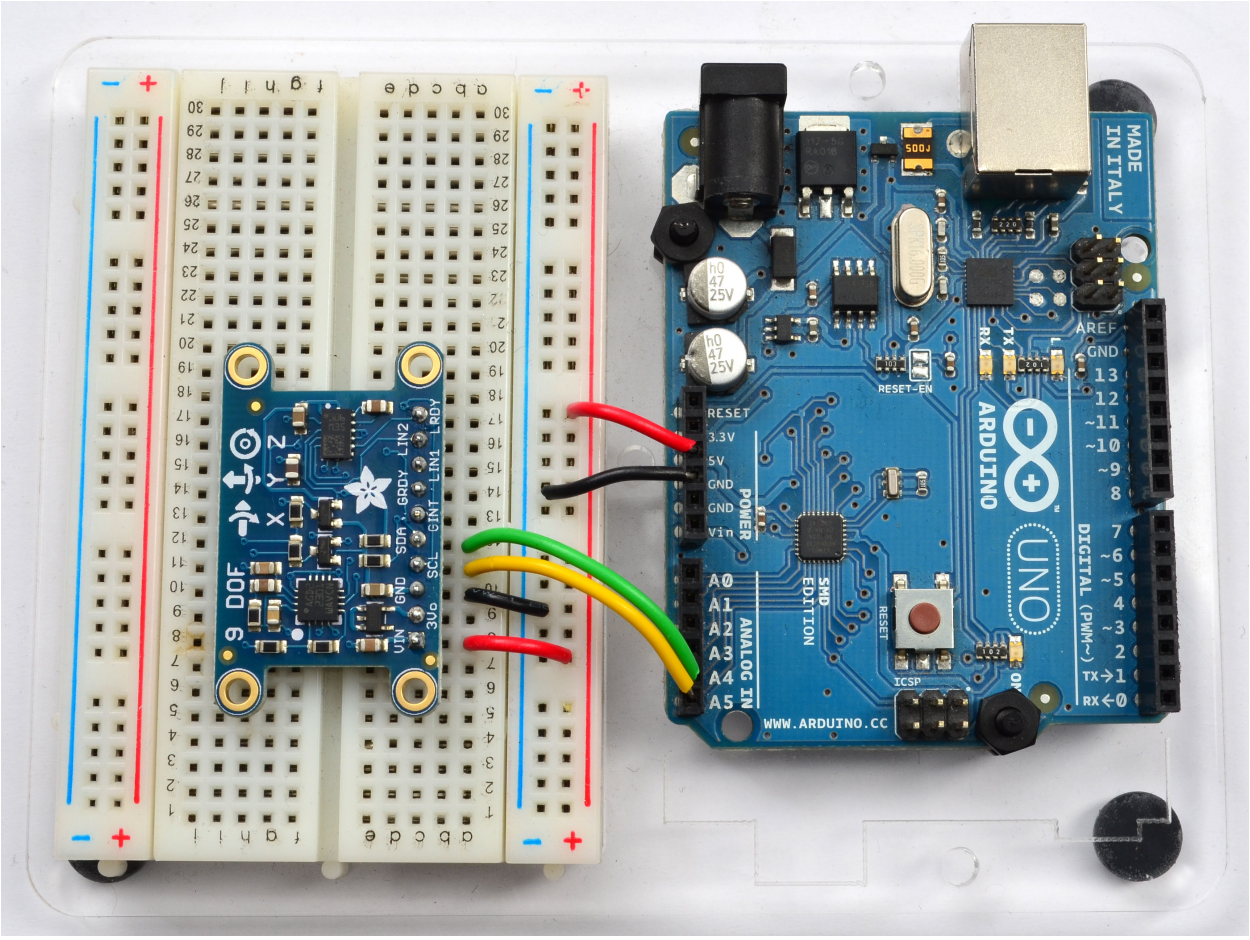


FIGURE 3
Adafruit 9DOF breakout board shown connected to Arduino Uno (Source: https://learn.adafruit.com/assets/14022)

In choosing a sensor to measure WBV, initial development was based on preliminary work done by Clark who had selected a nine degree-of-freedom (DOF) breakout board produced by Adafruit Industries (Figure 3). This particular board combines a LSM303DLHC chip, which provided a 3-axis accelerometer (±16g) and a 3-axis magnetometer (±8.1 gauss), with a L3GD20 chip as a 3-axis gyroscope (±2000dps). Data sheets for the LSM303DLHC and L3GD20 can be found in Appendix B and Appendix C respectively. The 9DOF sensors utilize the I2C protocol to communicate with each other and the Arduino board.

The Arduino platform comes with a free integrated/interactive development environment (IDE) (which can be downloaded from: http://arduino.cc/en/main/software) which allows users to write programs (referred to as sketches) to upload to and ultimately control an Arduino board. The programming language is based on the open-source Wiring framework and is implemented using C/C++ syntaxes. During the development process, an Arduino sketch (dof9.ino) was written using this IDE which output the acceleration in three axes and orientation (roll, pitch, heading) of the breakout board. This sketch can be found in Appendix D.

FIGURE 4
Adafruit CC3000 WiFi Shield mounted on an Arduino Uno (Source: https://learn.adafruit.com/assets/10740)

To enable wireless communication, several peripheral shields were tested to integrate with the 9DOF sensor. The first shield tested was the Adafruit CC3000 WiFi Shield shown in

Figure 4. The shield had similar issues to the NI cDAQ 9191 in regards to connecting to the OSU Medical Center network so the wireless hotspot capability of a Samsung Galaxy S3 was utilized to connect to the Sprint cellular network. After connecting to the internet, the next challenge was to collect data over the web which proved to be another challenge. After some research into methods to accomplish this task, use of the site Xively (www.xively.com) was tested. Xively is a platform to connect and manage the Internet of Things (IoT). There are several examples and applications of using the CC3000, Arduino, and Xively to monitor sensors and collect data remotely. After writing an Arduino sketch which transferred the acceleration and orientation of the 9DOF break out board, I discovered that this particular setup is better suited for low frequency measurement applications such as monitoring temperature or humidity per minute or hour. This was not a viable way to achieve the high sampling rate required for vibration measurements. The Xively_9dof_cc3300_v1.ino sketch can be found in Appendix E.

The next method tested to achieve wireless data transfer was XBee RF radios (Figure 5) by Digi (www.digi.com/xbee/). As described by Digi, XBee radios are based on the ZigBee standard and use the IEEE 802.15.4 networking protocol to achieve fast point-to-multipoint or peer-to-peer networking. They are designed for high-throughput applications requiring low latency and predictable communication timing [10]. They have many uses and applications in the IoT and are convenient because rather than sending or connecting to a external network, XBee modules are capable of creating ad-hoc networks on the fly and pass information back and forth to other radios.

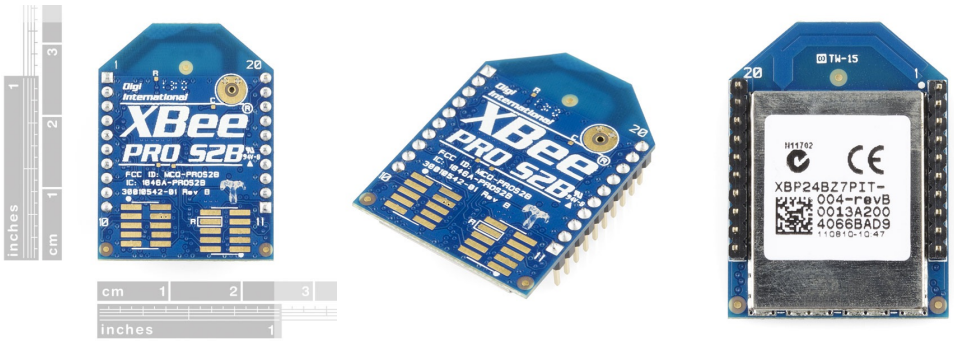


FIGURE 5
XBee Pro 63mW PCB Antenna - Series 2B (ZigBee Mesh)
(Source: https://www.sparkfun.com/products/10418)

Each XBee can be configured in one of three ways: 1) a Coordinator radio which forms, defines, and and manages the network, 2) a Router radio which can join networks as well as sends, receives, and routes information (useful for sending data over long distances), or 3) an End Device which can join networks and send/receive information but requires a Router or Coordinator as a parent device [11]. Networks are set up as one of four ZigBee topologies, illustrated in Figure 6, which each can have any number of end devices and routers but only one coordinator to act as a central hub.

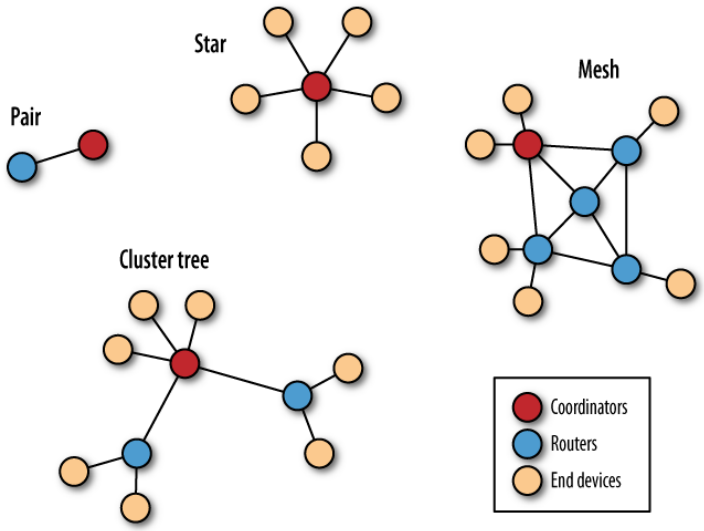


FIGURE 6
ZigBee topologies (Pair, Star, Cluster Tree, and Mesh) [11]

Using an XBee shield for the Arduino Uno, one can connect multiple 9DOF sensors each to an XBee end device node. This would then create a sensor network which can send the acceleration and orientation data wirelessly to a coordinator radio connected to a host computer. Of the many configurations of XBee radios, the XBee Pro 63mW PCB Antenna - Series 2B model was chosen for its 1 mile range (over the 300ft and 400 ft range of the non Pro Series 1 and series 2 models respectively) and greater flexibility for manual configuration using X-CTU software by Digi (which can be downloaded from: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/xctu).

The details of the configuration of the XBee network are detailed in a later section, however, through the system development process, a significant issue arose. The XBee radios transfer and receive data through serial communication using highly formatted API frame packets. This API protocol works well with transmitting raw, analog data but not data that has been partially processed using another type of information formatting.

The I2C protocol for the 9DOF chip treated the data in such a way that it was not compatible with and interfered with the ZigBee protocol. The transferred data came through corrupted and bytes of information were lost in transmission. This proved to be unsuitable for this study. Therefore, it was decided to discontinue use of the Adafruit 9DOF breakout board and an analog accelerometer was selected as the primary measurement device. An analog sensor requires no processing or conditioning of the data, other than simple mapping of the voltage range to acceleration limits, before it is received by a computer and upon testing this proved to resolve the issue of lost bytes and corrupted data.

Initially, the ADXL326 3-axis MEMS accelerometer (±16g) was selected but upon testing on a PWC it proved to be too wide a range and provided poor resolution of the data. Ultimately, the ADXL335 3-axis MEMS accelerometer (±3g) was selected which is described as "perfect for high-resolution static acceleration measurements such as tilt-sensing, as well as for moderate dynamic accelerations from motion, shock or vibration" [12]. It comes as a breakout board from Adafruit Industries which feature 3.3V voltage regulation to easily interface with the Arduino. The data sheet for the ADXL335 can be found in Appendix F.

One final component that was integrated with the Arduino was an Adafruit data logging shield (Figure 7). This shield offers an SD card interface to store large amounts of data locally. This was initially used as a safety measure to ensure that data and, more importantly, time was not lost in the event that an XBee radio would lose connection to the its coordinator or a wire was knocked loose. It became the primary method for collecting data after encountering development issues with XBee data transmission which will be discussed in a later section. It was desirable to avoid collecting unusable data and having to rerun trials as much as possible. This required the use of a SD card with a high write speed so a Class 10 card with 30Mb/s read/ write speed was selected.

Table 1 shows a summary of selected modules for data collection.

FIGURE 7
Adafruit data logging shield with inserted SD card mounted on Arduino Uno
(Source: https://learn.adafruit.com/assets/7024)

TABLE 1
Data Collection Component Summary

Module	Purpose
ADXL335 - 5V triple-axis accelerometer (+-3g analog out)	Acceleration measurement
XBee Pro 63mW PCB Antenna - Series 2B (ZigBee Mesh)	Wireless data transfer
Adafruit Assembled Data Logging Shield for Arduino	Interface for data storage
Arduino Uno - R3	Modular Controller
SparkFun XBee Shield	Interface for wireless radio
SanDisk 8GB SD Card - Class 10 (30Mb/s)	Local data storage

2.4 Software Development

Four programming interfaces were used to complete this project:

1. **X-CTU**: Used to configure XBee radios. Sets XBee role (Coordinator, Router, End device) and establishes network settings.
2. **Arduino IDE**: Used to control the Arduino Uno and attached shields. Programs are known as Sketches. Each sketch is uploaded to an Arduino via USB.
3. **Processing**: Used to create data or sensor based animations. Used to plot transmitted XBee data on a remote computer screen.
4. **MATLab**: Used to post-process data, run calculations, and produce static graphs and charts.

2.4.1 X-CTU Development

The first step was to configure the XBee radios to allow them to create a network. X-CTU is an open source program made by Digi International Inc. to set up an XBee network and is available for free from: http://www.digi.com/support/productdetail?pid=3352&type=utilities. The interface is shown in Figure 8 with the coordinator radio attached to the computer via USB. *Building Wireless Sensor Networks: A Practical Guide to the ZigBee Mesh Networking Protocol* by Robert Faludi was the primary reference for learning and understanding the configuration process. It should be referenced for a detailed walkthrough of the set up process. Key notes and settings are shown in Table 2 while full configuration profiles for the coordinator and router radios can be found in Appendix G and Appendix H respectively.

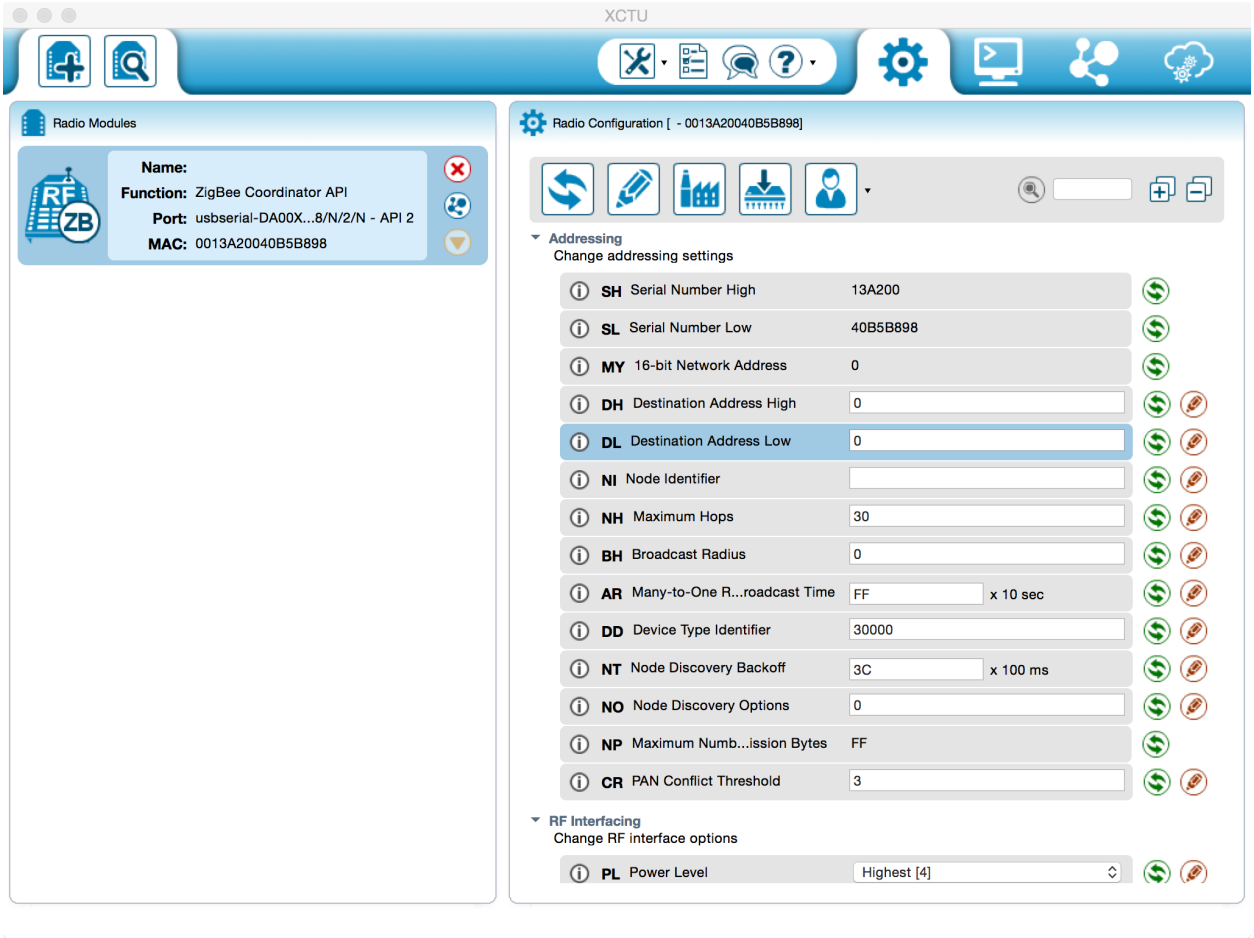


FIGURE 8
X-CTU configuration screen for XBee coordinator radio

TABLE 2
XBee Configuration Notes

AT	Description	Coord. Value	Router Value
	• Ensure coordinator radio is configured as a ZigBee Coordinator API with matching firmware • Ensure router radios are configured as ZigBee Router AT with matching firmware • Label radios with tape to identify them		
ID	Ensure each radio in the network has the same Pan ID	2001	2001
SL	Serial Address Low: Take note of this value for each radio. Used to identify each radio from another. Can also be found on the back of each XBee under 0013A200	40B5B898	40B554DD 40B554B1 40B5B89F
DH	Destination Address High: For router radios,	0	0
DL	Destination Address Low: For router radios,	0	0
BD	Baud Rate: Keep this consistent through all code in all programs that interface with serial data	115200 [7]	115200 [7]
SB	Stop Bits: Two stop bits were used in this study	2 [1]	2 [1]
AP	API Enable: Ensure API Mode is set to 2 for coordinator radio	2	
D0	AD0/DIO0 Configuration: Set the router radios to ADC [2] and leave coordinator Disabled [0]	0	2
D1	AD1/DIO1 Configuration: Set the router radios to ADC [2] and leave coordinator Disabled [0]	0	2
D2	AD2/DIO2 Configuration: Set the router radios to ADC [2] and leave coordinator Disabled [0]	0	2
IR	I/O Sampling Rate: Ensure this remains consistent for all radios (in hex so 32 = 50 ms)	32	32

The API Console, shown in Figure 9, demonstrates how acceleration data is transferred between XBees. Data is sent in highly structured packets called frames. These frames provide a wealth of information including the address of the sender radio, the length of the frame, the

number of samples included in the frame and the actual data that was transmitted. XBee sends these frames in hexadecimal notation serially. A sample of this frame data looks like:

```
      <frame>
        <frame_index>12</frame_index>
        <frame_date>1428484389522</frame_date>
        <frame_mode>received</frame_mode>
        <frame_op_mode>api2</frame_op_mode>
        <frame_payload>7E001692007D33A20040B554DD097501010000070200 03F0019C79</
frame_payload>
      </frame>
```

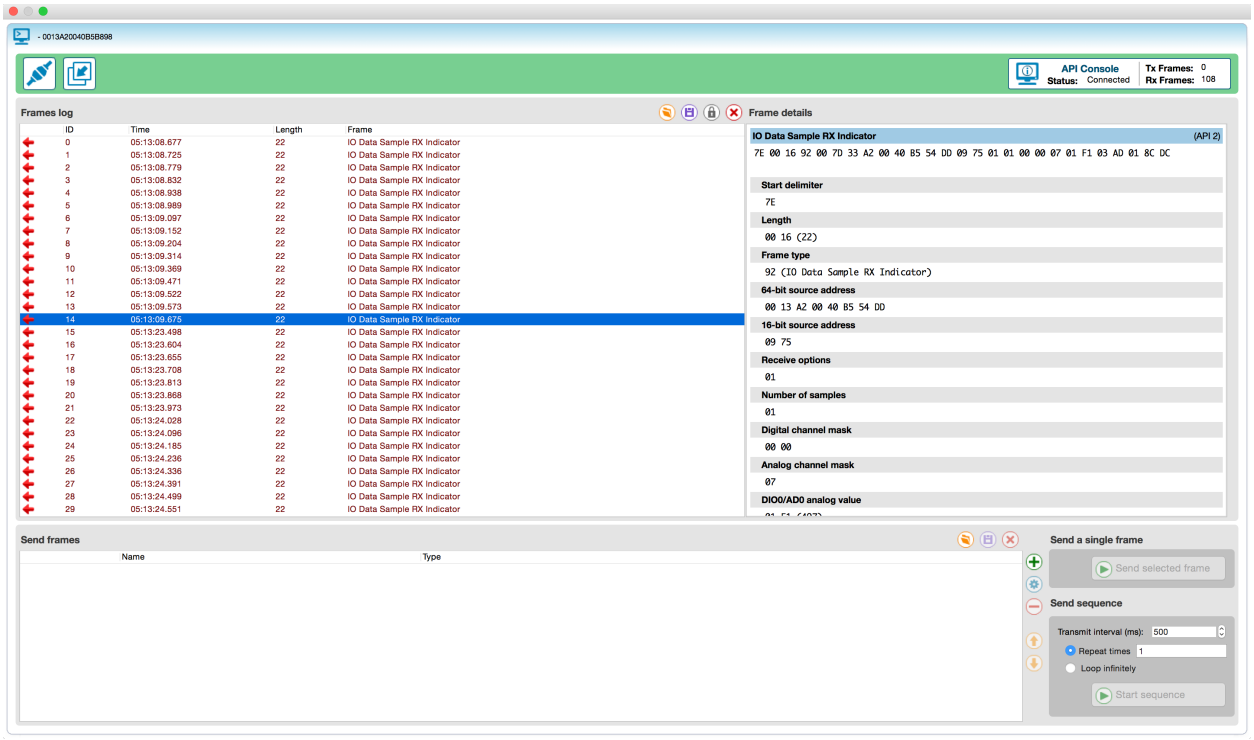


FIGURE 9

X-CTU API Console showing received frame data from a remote XBee wired to an accelerometer

By understanding the order and structure of the informations in the frame, the string of numbers and letters can be translated to the following:

IO Data Sample RX Indicator (API 2)

7E 00 16 92 00 7D 33 A2 00 40 B5 54 DD 09 75 01 01 00 00 07 02 00 03 F0 01 9C 79

Start delimiter: 7E

Length: 00 16 (22)

Frame type: 92 (IO Data Sample RX Indicator)

64-bit source address: 00 13 A2 00 40 B5 54 DD

16-bit source address: 09 75

Receive options: 01

Number of samples: 01

Digital channel mask: 00 00

Analog channel mask: 07

DIO0/AD0 analog value: 02 00 (512)

DIO1/AD1 analog value: 03 F0 (1008)

DIO2/AD2 analog value: 01 9C (412)

Checksum: 79

Using the structured format of API frames like the one above, programs were written using the Arduino and Processing environments which were capable of producing meaningful data for this study.

2.4.2 Arduino Development

Two Arduino sketches were required for wireless data transmission over XBee: one to use on the Arduino controlling each of the three sensor nodes attracted to the PWC and another for an Arduino connected to the coordinator XBee and a computer for receiving and displaying data. The sketch on the sensor node used the Arduino's analog inputs A0, A1, and A2 for receiving data directly from the x-,y-,z-axes of the ADXL335. The data was then sent to the Arduino's digital output pins which were capable of pulse width modulation (PWM) which were

wired to the XBee for transmission. The sketch also converted the raw values from the analog input pins to acceleration in meters per second. These calculated values were then logged with the raw data to the SD card as a comma separated value (CSV) file or had the option to be written to the serial monitor for calibration purposes as shown in Figure 10. The full sketch of pwcspZBSensorNode.ino can be found in Appendix I.

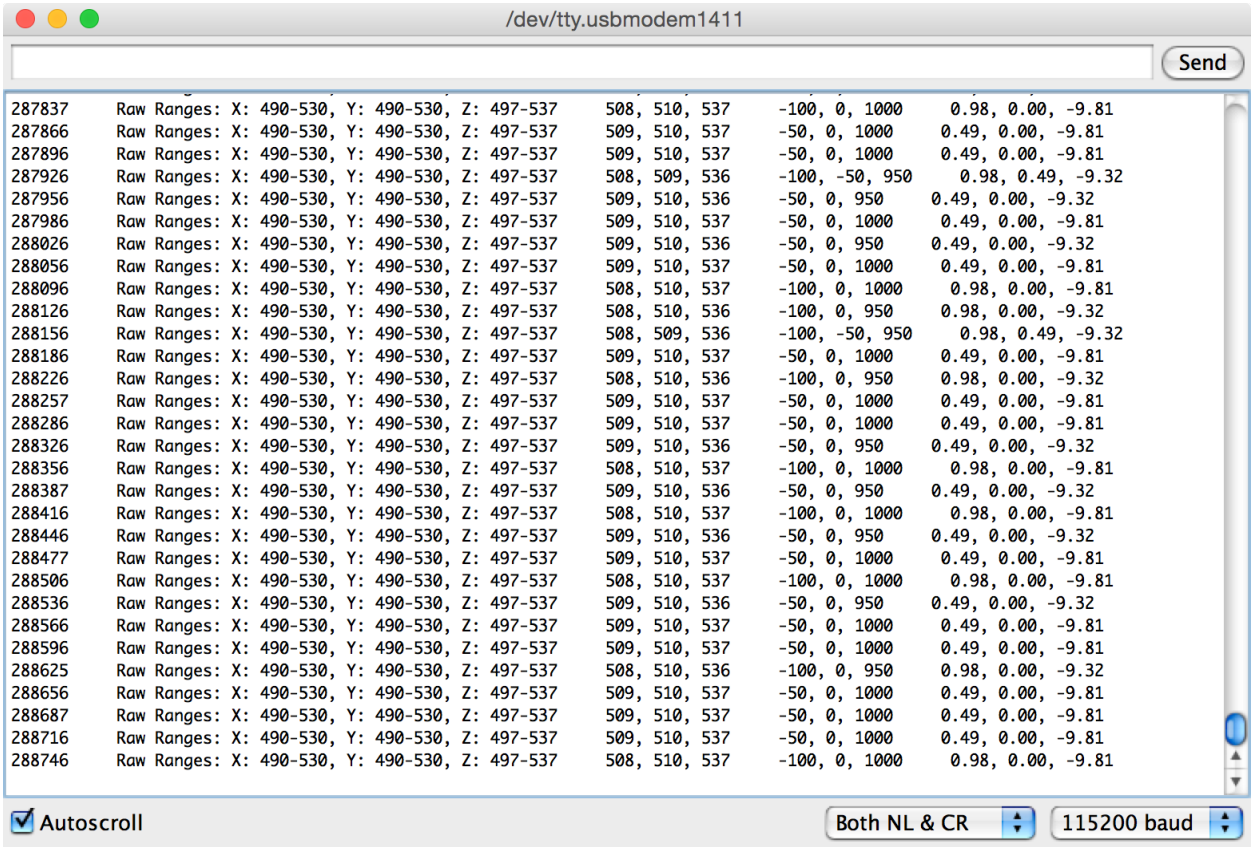


FIGURE 10
Serial monitor readout of Sensor Node acceleration data for calibration

There were some issues in obtaining suitable sampling rates using the initial code. To interface with the SD card, the sketch requires inputs for the sample size (in order to reduce noise in the data), log interval (milliseconds between entries i.e. sampling rate), and synch interval (milliseconds before writing the collected data to the disk). After the initial set of data collection

runs, it was discovered that the resulting sampling rate was too low to produce accurate frequency readings. As will be discussed later, this would not have been problematic since all of the analysis takes place in the time domain on acceleration amplitude. However, following the ISO standard for WBV studies, the time series data is weighted based on frequency its content. Sampling frequencies below a Nyquist frequency of 160 Hz (for evaluating vibration between 05. and 80 Hz) would produce inaccurate frequencies and thereby disrupt the weighted acceleration magnitudes.

This error was discovered after the bulk of the data was collected. Action was taken to increase the sampling rate by first educing the amount of data written to only time values and the acceleration of the three axes. Furthermore, the sample size, log interval, and synch interval variable values were toggled in the Arduino sketch. Coding to enable functionality for the XBee shield was removed as well and only code to write to the SD card was included. This updated sketch, pwcspSDSensorNode.ino, can be found in Appendix J. Upon rerunning tests, a minimum sampling rate of 144.6 Hz was able to be achieved. This covered a range of WBV from 0.5 to 72.3 Hz but does not accurately weigh frequencies above 72.3 Hz. Due to time constraints, further adjustment and retesting was not possible, however, both sets of data are included in this paper and the results, limitations, and suggested next steps will be discussed in Chapter 4. The initial data set using pwcspZBSensorNode.ino will be discussed as Data Set 04 while the rerun data using pwcspSDSensorNode.ino will be referred to in this paper as Data Set 08.

The base station was not used to collect data, however, the sketch for the coordinator arduino decoded API frames received from the sensor node XBees. These frames included information about the identity of the sender XBee and the analog acceleration data from each of

the three axes. This information could either be printed to a serial monitor or read by Processing to graph the incoming data. The full sketch of pwcspZBBaseStation.ino can be found in Appendix K.

2.4.2 Processing Development

Processing was used in an attempt to create a visual realtime plot of all of sensor data received by the base station. The full sketch used in the development process, pwcspMiltiSensorPlot.pde, can be viewed in Appendix L. The sketch takes ASCII-encoded strings of data from the serial port and then graphs them. Through the development process, the sketch is able to produce a window divided into three horizontal sections (one for each of the three accelerometers on the PWC) and each with three horizontal subsections (one for each of the three axes). The widow displays an abbreviated name for up to three nodal XBee radios and plots each data point for the acceleration amplitude data on a vertical axis. Each point is mapped horizontally based on the time it was collected. Each axis is plotted in a different color. When any single plot reaches the right end of the display window, the data clears for that subsection and begins again at the left edge of the display window.

The processing sketch was not used due to interruptions that occurred in the data being displayed for any one axis. It was not clear at what point in the transmission or processing system the interruptions were occurring. It has not yet been discovered whether this was due to signal losses over the XBee radios, or problems at the serial port, or in the Processing sketch itself.

The difficulties encountered with making the measurement system wireless in the sense of over-the-air transmission was delaying actual data collection from commencing. Because of

this and the limited time available for this phase of the study, it was decided to cease troubleshooting and development of tXBee data transmission and rely solely on the SD card to collect and store data.

Chapter 3: Whole Body Vibration Testing

3.1 Introduction

The objective of this phase of the study was to model the dynamic characteristics of the power wheelchair by determining the transmissibility of whole body vibration through multiple points of interest. To meet this objective, a procedure was developed using the measurement system described in Chapter 2. This procedure will later be used to test several wheelchair models against each other; however, for this initial development phase, one PWC was used: the Invacare TDX SP.

3.2 Instrumentation

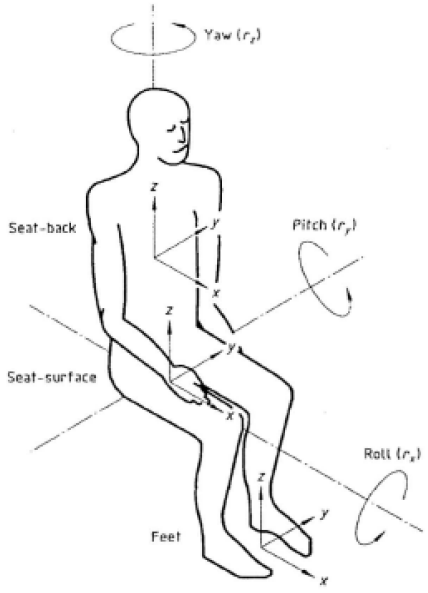


FIGURE 11
Basicentric coordinate system as defined by ISO 2361-1

The first step was selecting three locations which would serve as our points of interest. The ISO 2361-1 standard, which defines methods for quantifying WBV in relation [14], indicates the seat-surface, seat back and feet as three reference points and basicentric coordinate systems for objects in contact with a seated individual as shown in Figure 11. When measuring WBV, the basicentric coordinate system treats the z-axis as the critical axis, parallel to the the length of the spine [15].

For this study, the foot plate was omitted as a measurement location and instead replaced with the PWC chassis frame. Typically; when assessing WBV for motor vehicles, industrial movers, and public transport vehicles; the floor, seat surface, and seat back all act as input sources of vibration. However, in the case of PWCs and this study, the foot plate is suspended from the seat surface and has no direct contact with the source of vibration. Furthermore, this

study sought to compare the transmission of WBV from a chosen source to multiple output POIs. Based on some preliminary tests and past literature, we determined that the seat surface and back support were the two most critical POIs to study and the chassis frame of the PWC was chosen as the source input.

Something to note is that because this methodology would later be used across several different wheel chair models, it was decided that measurements be taken at the seat pan and back plate underneath the surface of the seat cushions. Multiple styles of seat cushions are available for any given PWC and are typically interchangeable. Measuring at the seat pan and back plate isolated the characteristics of the chair and removed from the data any variations on vibration reduction which the cushions and pads would provide. Previous work by DiGiovine et. al. [16] and Garcia-Mendez et. al. [17] has detailed WBV transmissibility for several models of seat cushions which can be referenced to make informed cushion selections.

To mount the sensor modules to the wheelchair, custom enclosures were designed using SolidWorks and 3D printed. A rendering of the SolidWorks models is shown below in Figure 12. The enclosures' primary purpose was to protect the electronics from damage-causing impacts and, for the Arduino stack, provide mounting points to affix them to the PWC. Mounting holes were drilled for each accelerometer at 1) the bottom-center of the seat pan (Figure 13.a), 2) the top-center of the back plate 55.5cm above the seat pan (Figure 13.b), and 3) the rear-center of the chassis frame 29 cm behind the axis of the middle wheel (Figure 13.c). Each accelerometer was mounted in a miniature breadboard which was then rigidly bolted to the enclosure and PWC.

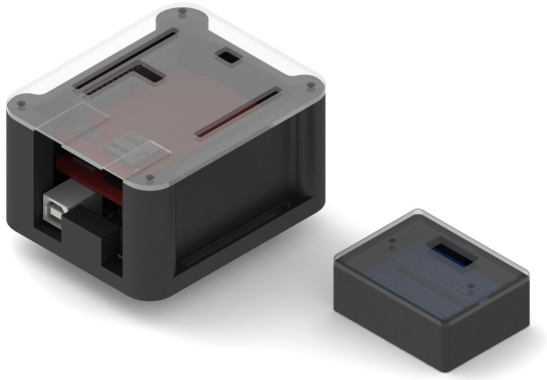
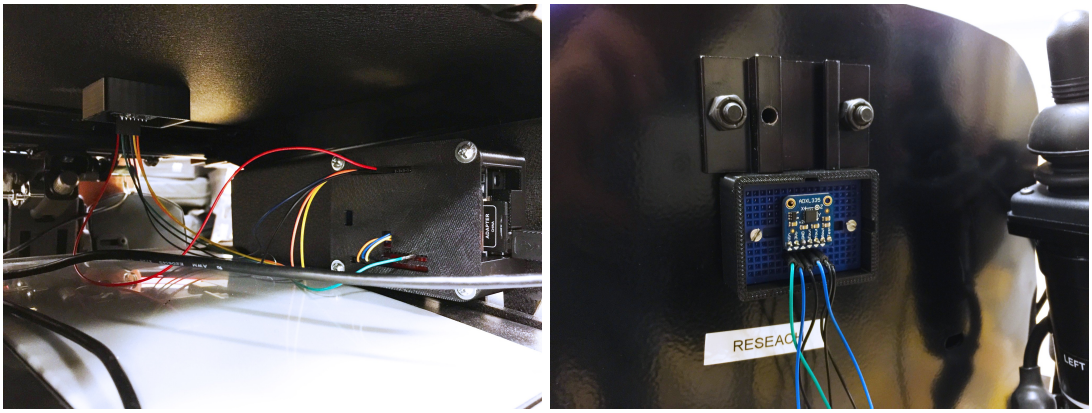
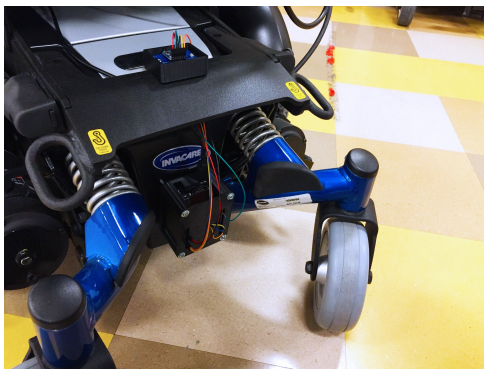


FIGURE 12
Sensor Node and Accelerometer Enclosure Rendering



(a) (b)



(c)

FIGURE 13
Sensor modules mounted at a) seat pan, b) back plate, c) chassis frame of the Invacare TDX SP

3.3 Testing Methodology

Testing to collect acceleration data was based on the Wheelchair Skills Test (WST). The WST is a part of the Wheelchair Skills Program (WSP) developed by Dalhousie University to train and asses wheelchair users. A list of the 30 skills tested in the WST are listed in Appendix M. Of these skills, the following were chosen to repeat for this study:

TABLE 3
Selected Wheelchair Skills Tested (*Indicates Reverse Direction)

Order	WST #	Individual Skill
1	7	Rolls forwards (10 m)
2	8	Rolls backwards (2 m)
3	27	Gets over threshold (2 cm)
4	11	Turns in place (180º)
5	11*	Turns in place (180º)*
6	24	Rolls across side slope (5º)
7	24*	Rolls across side slope (5º)*
8	25	Rolls on soft surface (2 m)
9	25*	Rolls on soft surface (2 m)*
10	20	Ascends 5º incline
11	23	Descends 10º incline
12	22	Ascends 10º incline
13	29	Descends low curb (5 cm)
14	N/A	Descends 7.5º incline
15	N/A	Ascends 7.5º incline
16	28	Ascends low curb (5 cm)
17	21	Descends 5º incline
18	N/A	Ascends low curb (3 cm)
19	26	Gets over gap (15 cm)
20	N/A	Descends low curb (3 cm)

Testing was conducted in the Assistive Technology Center at The Ohio State University's Wexner Medical Center, where a physical course (Figure 14) was set up with the events shown in Table 3.

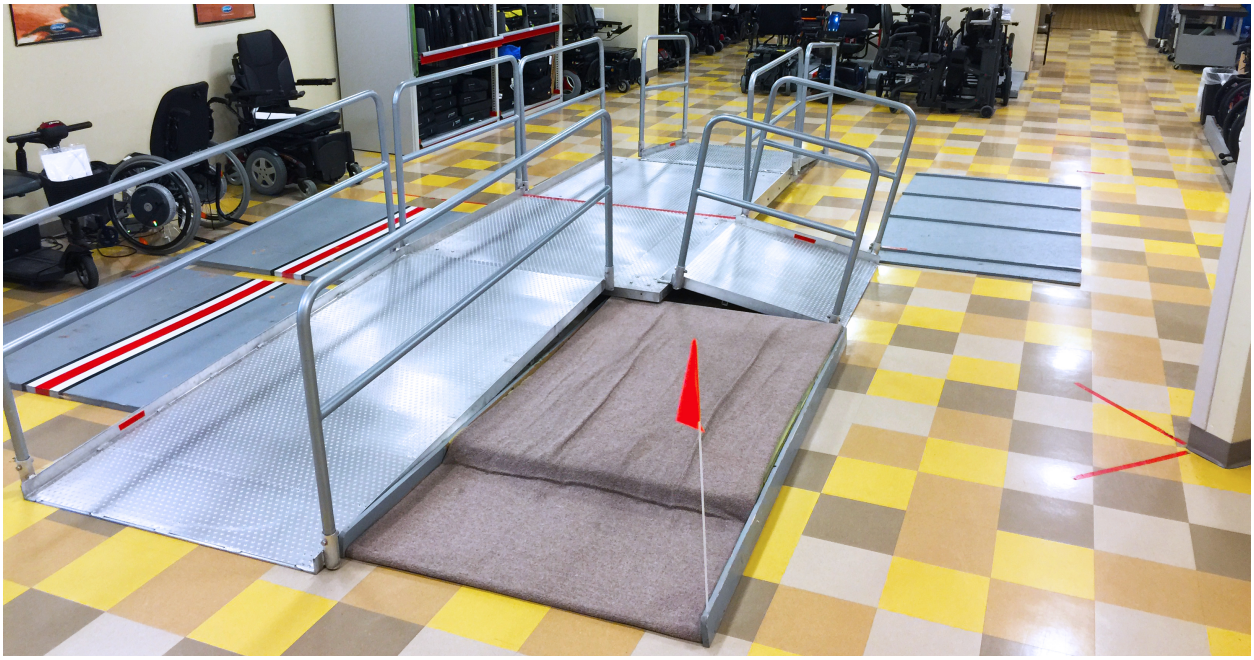


FIGURE 14
Wheelchair Skills Course at the Assistive Technology Center

An anthropomorphic test device (ATD) was utilized for the initial data collection (Data Set 04) to simulate a person occupying the PWC. The ATD, shown in Figure 15, was a 50th percentile male Hybrid III (otherwise known as “midsize male”) with a whole body mass of 78.20 kg. It was seated upright in the PWC and secured using the provided waist seatbelt and torso strap. The ATD loaded the system which provided the necessary mass to give more accurate WBV measurements. The PWC was fitted with an attendant joystick at the rear which allowed the tester to control the motion of the chair and navigate through the course. For DataSet 08 the ATD was replaced by a 100 kg wheelchair test device (WTD). The WTD can be seen in Figure 16.b. The total mass of the PWC, WTD, and instrumentation totaled 240.1 kg.

Before testing, a GoPro Hero 3+ Black Edition camera was fitted to the head of the ATD using a head band mount (not used for Data Set 08). This camera recorded a first hand point of view (POV) for each run and gave investigators a look at what an operator would see while

completing course obstacles. The recorded video also served as a reference for any anomalies or variations that the PWC may have experienced between runs. The GoPro camera can be seen in its mount in Figure 15. For selected runs, recorded video was also captured a birds eye POV in real time using the GoPro and a third person POV using an iPhone 6+ recording in slow motion at 240 fps. These views captured the full body of the PWC and provided additional visual information on the overall motion of the ATD/WTD and PWC. Samples of these views are shown in Figure 16.

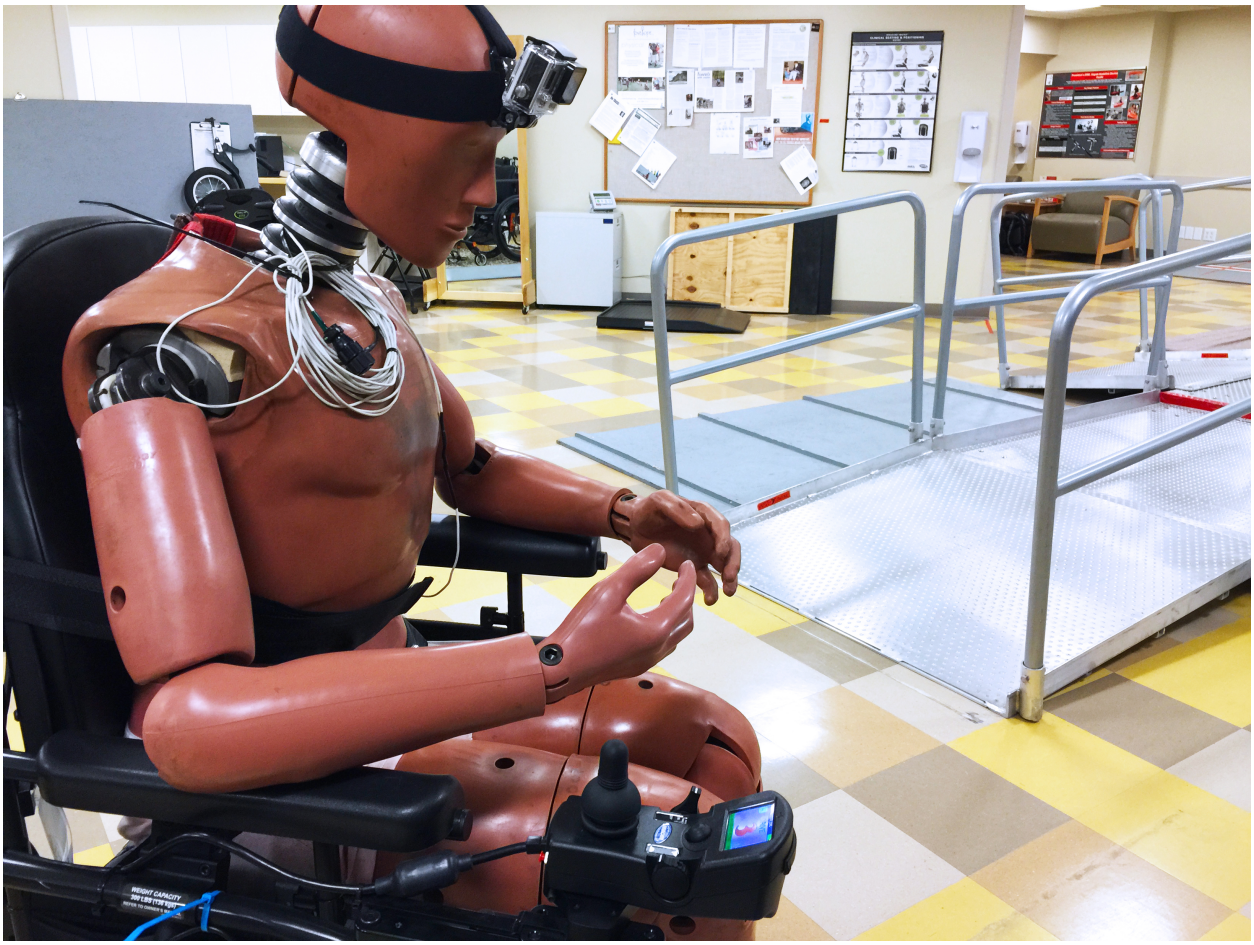


FIGURE 15
Anthropomorphic test device with mounted GoPro camera; seated in PWC

(a) (b)

FIGURE 16
Captured views: (a) third person POV by iPhone 6+ and (b) birds eye POV by GoPro

3.4 Test Procedure

Each run began by turning on the wheelchair and setting it to drive mode 3 (SDEEP_LVL). This drive mode equated to a speed of .88 m/s with the WTD occupying the PWC. Using the attendant joystick, the chair was moved to a designated starting position. The GoPro Hero 3+ was turned on and recording was started to capture the remaining initialization. Two battery packs, which provided power to the Arduino sensor modules, were turned on. This initialized their preloaded code and data collection to the SD cards would begin. This process took a few seconds so a brief pause was given and a visual check of the Arduinos' pin 13 LED would take place to ensure measurements were being recorded. When ready, the tester then took a rubber mallet and struck the frame of the PWC at the front tie down twice at roughly a 45 degree angle to the horizontal in the basicentric y-z and x-z planes. The consecutive strikes occurred approximately one second apart. This provided two clear spikes in the data for all sensors in all axes, giving clear reference points in time indicating the beginning of the run.

The tester then navigated the PWC and ATD through the chosen events from the WST listed in Table 3, allowing for a 3-5 second pause before and after each event. This was done for

two purposes, 1) to allow for any vibrations generated by the event or traveling motion to attenuate, and 2) to provide a clear space between each event to make identification of events on a time plot of the data simpler to decipher.

Following the final event, the tester took the rubber mallet and struck the frame of the PWC at the tie down location twice in the same manner as the start of the run. This provided two clear spikes in the data for all sensors in all axes, giving clear reference points in time indicating the end of the run. The USB cables connecting the Arduinos to their battery pack power supply were unplugged cutting off power and stopping data collection (the battery packs did not posses a simple off switch to cut off power so unplugging the sensor modules was the alternative option). Finally, the GoPro recording was stopped. This was the procedure for collecting acceleration data from the Wheelchair Skills Course.

3.5 Data Import and Processing

A complete publishing of the code used to process the collected data can be found in Appendix N along with the figures and values which it output. This section contains an overview of the process. An identical process was used for both Data Set 04 and Data Set 08. Published code for Data Set 8 can be found in Appendix O.

Multiple sampling runs were conducted. After a run was complete, the tester removed the SD cards from each of the three sensor modules and saved the logged csv file to a computer running MATLab. File sizes were roughly 1.4Mb. Files were renamed as ds#_location_axis: the # identified the which sampling run out of the series the data was obtained from; the location identified the measurement location as the back plate, seat pan, or chassis frame; and the axis

indicated the x-, y-, or z-axis measured. For example, the filename ds4_back_z indicated the z-axis of the back plate from data set 4.

Using MATLab's import tool, data was pulled in as column vector variables under this naming convention. An offset was then applied to each vector to counter minor calibration offsets from the accelerometers. Each axis was then assigned to a new name to correspond to the basicentric coordinate system explained in Section 3.2. Plots (Figure 17) were generated to show the adjusted raw data for each accelerometer.

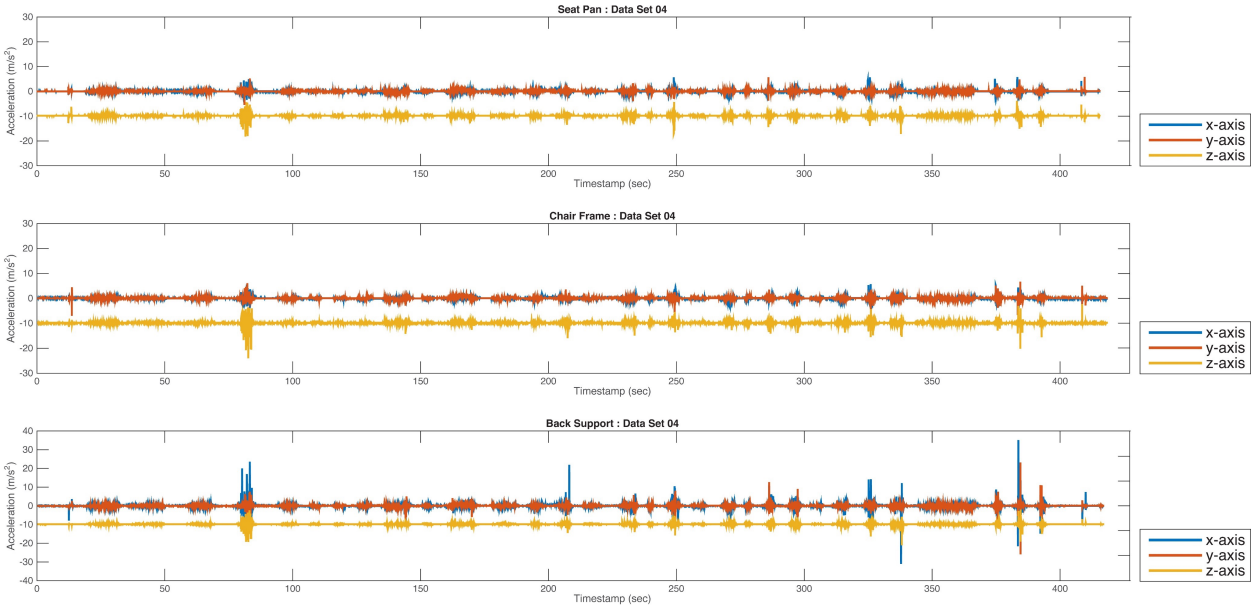


FIGURE 17
(Top) Seat Pan, (Middle) Chassis Frame, and (Bottom) Back Support accelerations for Data Set 04

The data was first plotted to allow the tester to obtain the times at which the initial and final mallet strikes occurred. These times were then written into the script to trim the data to include everything collected between these mallet events. The script then weighted frequencies for each of the nine axes (three coordinate axes x 3 locations) per ISO 2631-1 using a series of frequency weightings shown in Appendix P. Each direction is applied a different weighting based

on an understanding that people are more sensitive to some vibrational frequencies than others (detailed in Table 4 for WBV as well as motion sickness and hand transmitted vibration even though the latter types are outside the scope of this study). The vibration is perceived and experienced at varying magnitudes depending on the direction and point of contact with a person. Mansfield indicates that, “frequency weightings are designed not to affect those frequencies where the body is most sensitive and to attenuate at those frequencies where the response of the body is less sensitive” [2]. Because the weightings do not amplify the the signal at any frequency, the magnitude of the weighted signal should be less than the magnitude of the measured signal. After calculating the magnitude of the frequency weighted acceleration signals, we were able to compare different locations on the PWC to obtain an indication of vibrational transmissibility; or simply the ratio of vibration magnitude at an output location to an input location. Also, comparing the magnitude of the weighted to unweighted signals provided a way to validate that frequency weightings were applied properly.

TABLE 4
Summary of Most Common Frequency Weightings Used for Analysis of Human Vibrations Signals [2]

Frequency Weighting	Application Area	Frequency Range	Direction
W_b	Whole-body	0.5-80 Hz	z-seat
W_c	Whole-body	0.5-80 Hz	x-backrest
W_d	Whole-body	0.5-80 Hz	x-seat y-seat
W_e	Whole-body	0.5-80 Hz	Rotation seat
W_f	Motion Sickness	0.1-0.5 Hz	z-vertical
W_h	Hand-transmitted	8-1,000 Hz	x-hand y-hand z-hand
W_k	Whole-body	0.5-80 Hz	z-seat

One way to represent the magnitude of a time-based signal is to calculate the mean amplitude over time. Vibration, though, is oscillatory in nature centering around a center value (let's say zero for a zeroed signal); therefore, a mean or average of a signal would result in positive values canceling out negative values resulting in a mean of zero. This method does not provide a useful way of representing the magnitude of the frequency weighted acceleration data. Two alternative representations are the root-mean-square (r.m.s.) and vibration dose value (VDV).

The r.m.s. acceleration serves as a representation of the mean for a vibration signal by squaring each value before calculating the mean then taking the square root. r.m.s. acceleration is measured in units of m/s^2. Mathematically, the r.m.s. acceleration is expressed as:

$$a_{w\,r.m.s.} = \sqrt{\frac{1}{T}\int_0^T a_w^2(t)\,dt} \qquad [1]$$

where, $a_{w\,r.m.s.}$ is the frequency weighted r.m.s. acceleration, T is the duration of the measurement, and $a_w(t)$ is the frequency weighted acceleration at time t [2] as noted in Equation 1. The MATLab script accomplishes this using the code:

```
awrms=0;
        awrms_run=zeros(n,1);
        for i=1:n
            awrms=awrms+aw(i)^2;
            awrms_run(i,1)=(awrms*dt/tim(i))^0.5;
        end
        awrms=awrms_run(n);
```

where, `awrms` is the scalar frequency weighted r.m.s. acceleration, `awrms_run` is the running frequency weighted r.m.s. acceleration vector,`n` is the scalar number of values in the signal, `aw` is the frequency weighted acceleration vector, `dt` is the scalar time increment, and `tim` is the time vector.

A major limitation of the r.m.s. is that it is relatively insensitive to shocks and jolts experienced in a signal. Figure 18 shows how the r.m.s. will quickly increase in the event of a shock, but the shock's influence tends to decay as time increases. For this reason, calculating the VDV is the preferred method for representing magnitude for vibration signals similar to the ones collected in this study, which are characterized by shock based events. To check if VDV was more applicable than the r.m.s. acceleration, the crest factor (CF) was calculated for each axis. The crest factor is a calculated ratio of peak acceleration to the r.m.s. weighted acceleration, mathematically expressed by Equation 2:

$$CF = \frac{\max(a_w(t))}{r.m.s.(a_w)} \qquad [2]$$

where a_w is the frequency weighted acceleration. The ISO 2631-1 standard states that "the r.m.s. method is extended to crest factors less than or equal to 9 [14]" while alternative procedures should be used for crest factors greater than nine. All measured signals had crest factors greater than nine, thus, VDV was used as the primary comparison method.

The vibration dose value is a method for indicating the magnitude of a vibration signal. Unlike the r.m.s., VDV is based on a fourth power relationship which results in a greater sensitivity to shock events. As time increases, the VDV continually increases as well even if the magnitude of acceleration remains constant or low which can be seen in Figure 18. VDV is measured in units of $m/s^{1.75}$ (or sometimes just VDV) and mathematically expressed as:

$$VDV = \sqrt[4]{\int_0^T a_w^4(t)\,dt} \qquad [3]$$

where, T is the duration of the measurement, and $a_w(t)$ is the frequency weighted acceleration at time t. The MATLab equivalent of Equation 3 accomplishes this using the code:

```
VDVw=0;
        VDVw_run=zeros(n,1);
        for i=1:n
            VDVw=VDVw+aw(i)^4;
            VDVw_run(i,1)=(VDVw*dt)^0.25;
        end
        VDVw=(VDVw*dt)^0.25;
```

where, `VDVw` is the scalar frequency weighted VDV, `VDVw_run` is the running frequency weighted VDV vector`,n` is the scalar number of values in the signal, `aw` is the frequency weighted acceleration vector, and `dt` is the scalar time increment.

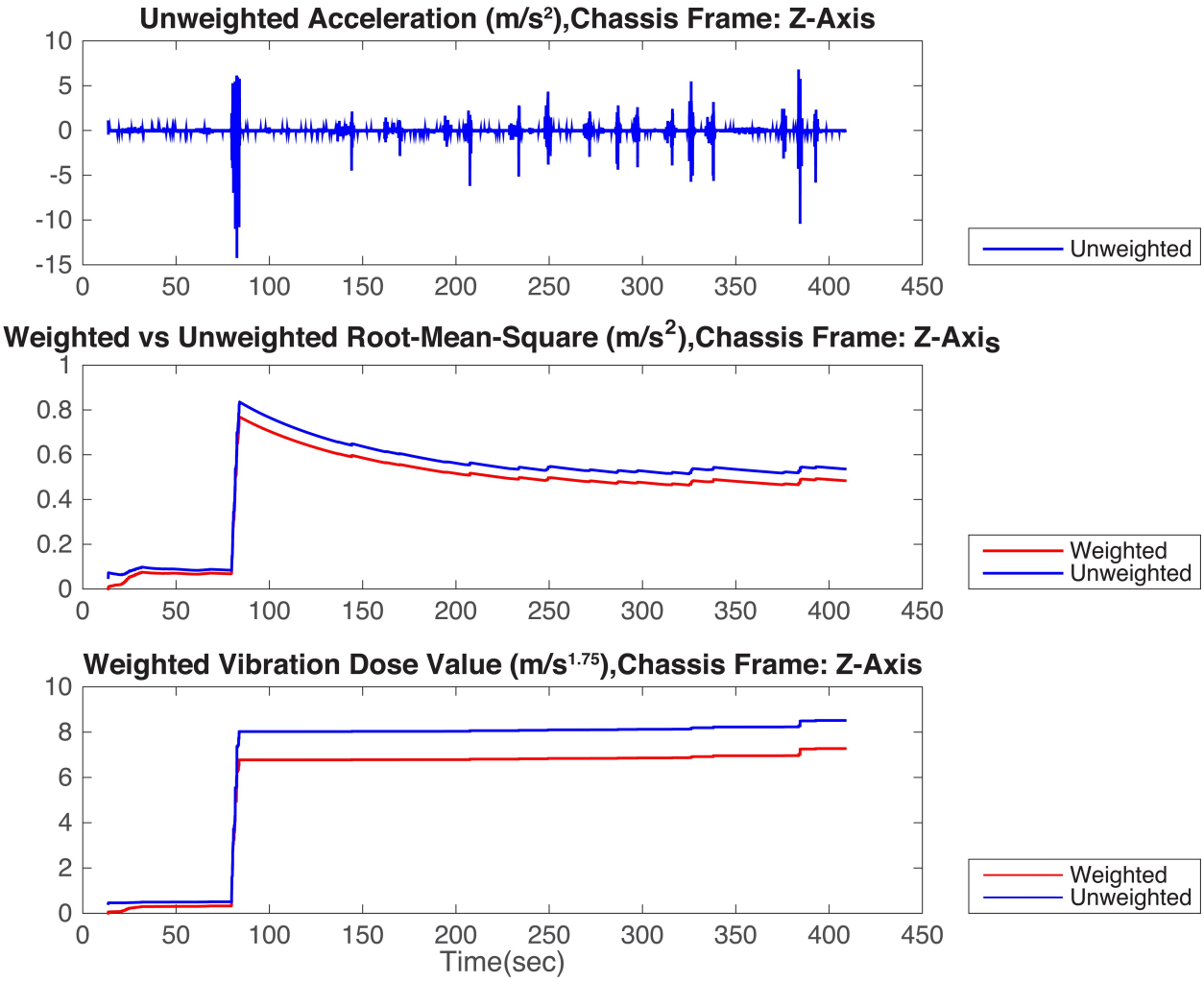


FIGURE 18

Data Set 04 weighted vs unweighted r.m.s. and VDV comparison for Chassis Frame in z-axis

Both the r.m.s. acceleration and VDV were calculated for both the weighted and unweighted signals for each of the nine measurement directions/locations. Lastly, a combined VDV was calculated to combine the axes at each POI. The combined VDV is calculated using Equation 4:

$$VDV_{xyz} = \sqrt[4]{k_x^4 VDV_x^4 + k_y^4 VDV_y^4 + k_z^4 VDV_z^4} \quad [4]$$

where the subscripts indicate the axis and k_x, k_y, and k_z are scaling factors applied to each axis. ISO 2631-1 details the values used as scaling factors for assessing health and comfort. These values along with a summary of the calculated data is shown in Table 5.

TABLE 5
Summary of calculated values for WBV analysis for Data Set 04

Location	Back Plate			Chassis Frame			Seat Pan		
Direction	**x-axis**	**y-axis**	**z-axis**	**x-axis**	**y-axis**	**z-axis**	**x-axis**	**y-axis**	**z-axis**
Frequency Weighting (0.5 - 80 Hz)	W_c	W_d	W_d	W_d	W_d	W_k	W_d	W_d	W_k
Scaling Factor (Health)	$k_x = 0.8$	$k_y = 0.5$	$k_z = 0.4$	$k_x = 1.4$	$k_y = 1.4$	$k_z = 1$	$k_x = 1.4$	$k_y = 1.4$	$k_z = 1$
Scaling Factor (Comfort)	$k_x = 0.8$	$k_y = 0.5$	$k_z = 0.4$	$k_x = 1$	$k_y = 1$	$k_z = 1$	$k_x = 1$	$k_y = 1$	$k_z = 1$
Unweighted RMS Acceleration [m/sec^2]	1.274	0.764	0.522	0.469	0.360	0.536	0.479	0.286	0.405
Weighted RMS Acceleration [m/sec^2]	1.218	0.337	0.286	0.275	0.187	0.484	0.309	0.127	0.371
Peak Acceleration [m/sec^2]	35.830	5.741	4.470	4.175	2.588	7.079	4.365	2.026	5.925
Unweighted VDV [m/sec$^{1.75}$]	20.540	14.000	7.811	4.678	4.640	8.516	4.857	3.680	5.999
Weighted VDV [m/sec$^{1.75}$]	20.340	4.363	3.776	3.251	2.058	7.272	3.688	1.578	5.382
Crest Factor (CF)	29.415	17.051	15.648	15.172	13.821	14.628	14.122	16.022	15.983
VDVw_{xyz}			20.358			7.355			5.665
VDVw_{xyz} (Health) [m/sec$^{1.75}$]			16.275			7.576			6.298
VDVw_{xyz} (Comfort) [m/sec$^{1.75}$]			16.275			7.355			5.665

The highlighted values shown in Table 5 provide a look into the transmissibility of WBV throughout the selected POIs for Data Set 04. One way to quantify transmissibility is through the seat effective amplitude transmissibility (SEAT) value (note: pronounced see-at). The SEAT value is defined by the dimensionless ratio of ride (dis)comfort at the output location to ride (dis)comfort at the input location. For this study, ride (dis)comfort was represented by the VDV therefore SEAT is expressed by:

$$SEAT\% = 100 \times \frac{VDV_{output}}{VDV_{input}}$$

This study treats the chassis frame location as the input source of vibration whereas the seat pan and back plate are outputs. Hence, SEAT was calculated using the following expressions:

$$SEAT\% = 100 \times \frac{VDV_{back\ plate}}{VDV_{chassis\ frame}}$$

$$SEAT\% = 100 \times \frac{VDV_{seat\ pan}}{VDV_{chassis\ frame}}$$

SEAT values were tabulated in Table 6 and Table 7 below. Combined VDVs used the health based axis scaling factor.

TABLE 6
Sample 04 SEAT% Values Chassis Frame to Back Support

Location	Chassis Frame			Back Plate		
Direction	**x-axis**	**y-axis**	**z-axis**	**x-axis**	**y-axis**	**z-axis**
VDV [m/sec$^{1.75}$]	3.251	2.058	7.272	20.340	4.363	3.776
VDV$_{xyz}$ [m/sec$^{1.75}$]	7.355			16.275		
SEAT% (Back:Frame)				625.65%	212.00%	51.93%
Combined SEAT%				276.8%		
Combined SEAT% (Health)				214.80%		
Combined SEAT% (Comfort)				221.27%		

TABLE 7
Sample 04 SEAT% Values Chassis Frame to Seat Pan

Location	Chassis Frame			Seat Pan		
Direction	**x-axis**	**y-axis**	**z-axis**	**x-axis**	**y-axis**	**z-axis**
VDV [m/sec$^{1.75}$]	3.251	2.058	7.272	3.688	1.578	5.382
VDV$_{xyz}$ [m/sec$^{1.75}$]	7.355			5.665		
SEAT% (Seat:Frame)				113.44%	76.68%	74.01%
Combined SEAT%				77.02%		
Combined SEAT% (Health)				83.10%		
Combined SEAT% (Comfort)				77.00%		

The tables below show the data calculated for Data Set 08 with Table 8, Table 9, and Table 10 echoing the information shown for Data Set 04 in Table 5, Table 6, and Table 7 respectively.

TABLE 8
Summary of calculated values for WBV analysis for Data Set 08

Location	Back Plate			Chassis Frame			Seat Pan		
Direction	**x-axis**	**y-axis**	**z-axis**	**x-axis**	**y-axis**	**z-axis**	**x-axis**	**y-axis**	**z-axis**
Frequency Weighting (0.5 - 80 Hz)	W_c	W_d	W_d	W_d	W_d	W_k	W_d	W_d	W_k
Scaling Factor (Health)	$k_x = 0.8$	$k_y = 0.5$	$k_z = 0.4$	$k_x = 1.4$	$k_y = 1.4$	$k_z = 1$	$k_x = 1.4$	$k_y = 1.4$	$k_z = 1$
Scaling Factor (Comfort)	$k_x = 0.8$	$k_y = 0.5$	$k_z = 0.4$	$k_x = 1$	$k_y = 1$	$k_z = 1$	$k_x = 1$	$k_y = 1$	$k_z = 1$
Unweighted RMS Acceleration [m/sec^2]	0.920	0.808	0.519	0.502	0.442	0.535	0.500	0.359	0.416
Weighted RMS Acceleration [m/sec^2]	0.669	0.333	0.303	0.282	0.214	0.474	0.291	0.136	0.367
Peak Acceleration [m/sec^2]	9.945	6.531	3.518	3.728	3.773	7.129	3.621	2.044	4.974
Unweighted VDV [m/sec$^{1.75}$]	10.670	11.540	8.080	4.810	6.472	8.230	4.708	4.992	5.653
Weighted VDV [m/sec$^{1.75}$]	7.415	3.966	3.747	2.855	2.607	7.496	2.928	1.344	4.773
Crest Factor (CF)	14.860	19.590	11.630	13.210	17.630	15.050	12.430	15.010	13.540
$VDVw_{xyz}$			7.674			7.562			4.941
$VDVw_{xyz}$ (Health) [m/sec$^{1.75}$]			5.957			7.740			5.341
$VDVw_{xyz}$ (Comfort) [m/sec$^{1.75}$]			5.957			7.562			4.941

TABLE 9
Sample 08 SEAT% Values Chassis Frame to Back Support

Location	Chassis Frame			Back Plate		
Direction	**x-axis**	**y-axis**	**z-axis**	**x-axis**	**y-axis**	**z-axis**
VDV [m/sec$^{1.75}$]	2.855	2.607	7.496	7.415	3.966	3.747
VDV$_{xyz}$ [m/sec$^{1.75}$]	7.5616			7.6740		
SEAT% (Back:Frame)				259.72%	152.13%	49.99%
Combined SEAT%				101.49%		
Combined SEAT% (Health)				76.96%		
Combined SEAT% (Comfort)				78.78%		

TABLE 10
Sample 08 SEAT% Values Chassis Frame to Seat Pan

Location	Chassis Frame			Seat Pan		
Direction	**x-axis**	**y-axis**	**z-axis**	**x-axis**	**y-axis**	**z-axis**
VDV [m/sec$^{1.75}$]	2.855	2.607	7.496	2.928	1.344	4.773
VDV$_{xyz}$ [m/sec$^{1.75}$]	7.5616			4.941		
SEAT% (Back:Frame)				102.56%	51.55%	63.67%
Combined SEAT%				65.34%		
Combined SEAT% (Health)				69.01%		
Combined SEAT% (Comfort)				65.34%		

Chapter 4: Discussion

SEAT% values equal to 100% indicate that the properties of the PWC have not increased or reduced ride comfort at the output location; values greater than 100% indicate that the comfort is worse at the output location than at the input source; values less than 100% indicate that the comfort at the output location is improved compared to that at the input source. The data from Data Set 04 indicates that 77% of WBV is transmitted from the chassis frame to the seat pan which can also be interpreted as a 23% overall reduction in vibration at the seat. However, along the x-axis, WBV levels were actually 13% greater than at the frame. The dynamic properties of the PWC result in greater discomfort along the x-axis. The data for the back plate indicates that only along the z-axis is WBV magnitude are reduced. The back plate experiences a 48% reduction in WBV magnitude from the frame along the z-axis; however, magnitudes are increased significantly along the x- and y-axes. According to the data, the back plate experiences over twice the level of WBV along the y-axis and over six times the WBV level along the x-axis compared to the frame. Combining axes results in an overall decrease in ride comfort at the back plate, the back being 2.1 times more uncomfortable than the chassis frame in regards to vibration.

Plotting the axial VDVs against each other provides a graphical comparison of the magnitude of WBV at each POI. These plots for Data Set 04 are shown in Figure 19, Figure 20, and Figure 21 for the x-axis, y-axis, and z-axis respectively.

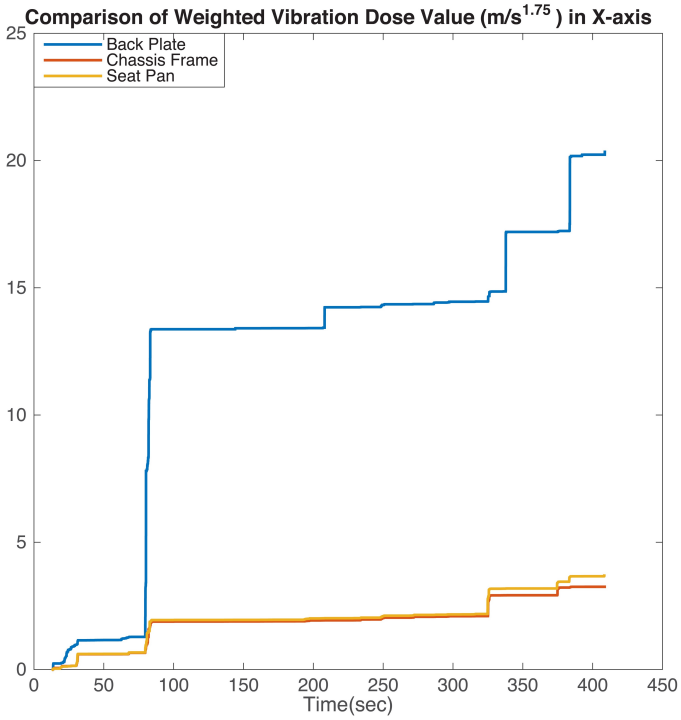


FIGURE 19
Comparison of Weighted VDV for all POIs in the x-axis for Data Set 04

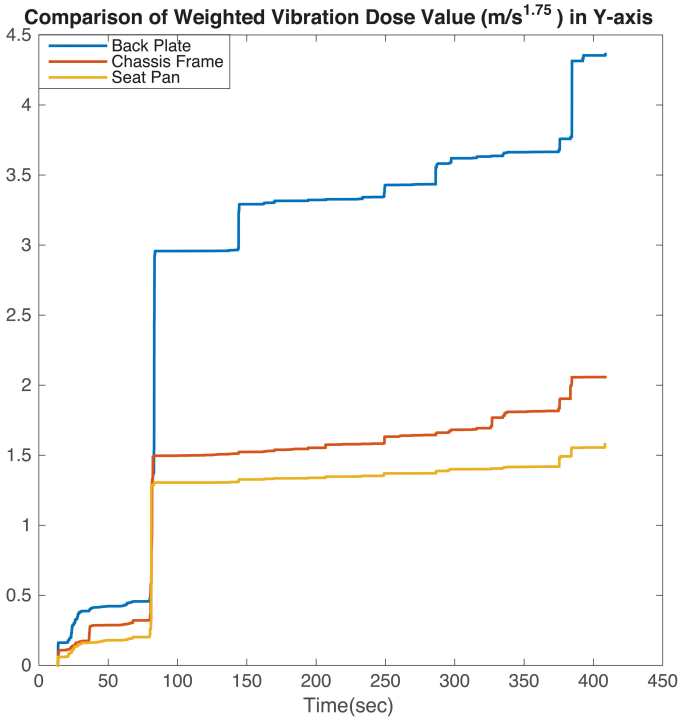


FIGURE 20
Comparison of Weighted VDV for all POIs in the y-axis for Data Set 04

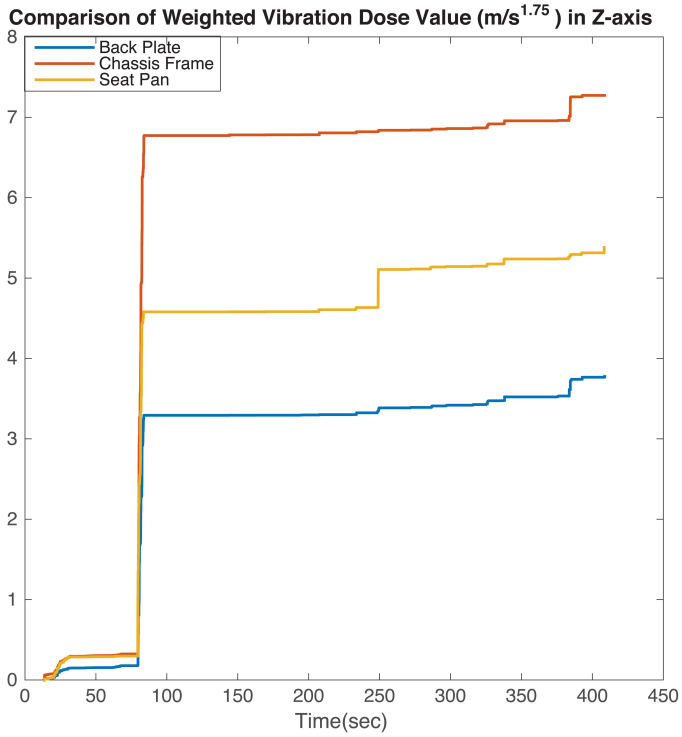


FIGURE 21
Comparison of Weighted VDV for all POIs in the z-axis for Data Set 04

Something of note can be interpreted by these plots. Typically, WBV analyses have primarily treated the z-axis at the seat as the primary POI and primary direction for these studies. Figure 21 graphically illustrates this axis and the VDV plots show a typical, expected relationship between the three locations; the source input has the largest magnitude or greatest discomfort level which is then reduced as the vibration travels away to the output locations, before the user experiences the motion. This matches the relationships to other vibrational waves which dissipate with an increase in distance away from the source. This relationship between the locations is also desirable because it indicates that the dynamic characteristics of the PWC effectively reduce shock and vibration along this axis.

Figure 19 and Figure 20 show a different relationship. The y-axis plot in Figure 20 shows the 2.1 times difference in VDV or comfort level between the back plate and the frame. The seat

pan still shows a reduction in WBV along the y-axis. Figure 19 shows the x-axis which is reverses the order desired order of the VDVs at each POIs. In this case, the source input has the smallest magnitude or least discomfort level which is then amplified as the vibration travels away to the output locations. This raises some interesting thoughts and considerations.

Perhaps PWC designs, development, and testing ought to reassess the importance of the back plate. Most WBV assessments have been for applications for persons who do not use a wheelchair and can typically adjust their posture to remain upright regardless of pitch. Conversely, persons who use PWC often have limited control of their torsos and therefore pitch, roll, and vibration along the x- and y-axes have a greater effect on them as they are not always capable of compensating or countering this motion.

Additionally, power wheelchairs, mid wheel drive PWCs in particular, have unique characteristics that set them apart from traditional devices which cause WBV such as cars and trains and industrial movers. The mid wheel drive makes the device behave like an inverted pendulum rather than a centrally balanced vehicle such as a four wheeled car. This was realized after analyzing some of the captured slow motion video. The affects of this characteristics can be seen in Figure 16.a as the PWC pitches forward upon ascending a ramp. As mass is located further along the vertical moment arm, away from this central axis of the mid wheel, the magnitude and effects of forces are amplified. This could explain why VDVs in the x-axis are significantly larger than current models may predict. The chassis accelerometer was located the closest to the mid wheel axis while the seat pan then the back plate accelerometer were located increasing distances away. This explains the trend seen in Figures 19 -21 as the VDV in the x-

and y- axis become more prominent as the POI location is further from the mid wheel axis. The data collected for Data Set 08 mirrors this trend.

Data Set 04 was processed with a low sampling rate and therefore the frequency content of the data was not properly weighted. The frequency weightings would have reduced the magnitude of some of the data. The weightings outlined in ISO 2361-1 do not amplify the signal at any frequency but diminishes frequencies that are less perceptible to humans. This is shown in Figure 22 for principle frequency weighting curves. Due to this, the amplitudes of the unweighted data may provide more useful information on the raw magnitudes of vibration measured due to all the analysis other than the weightings being completed in the time domain. The unweighted data should not be used to accurately asses the perception, health, or comfort of a user according to the ISO standard.

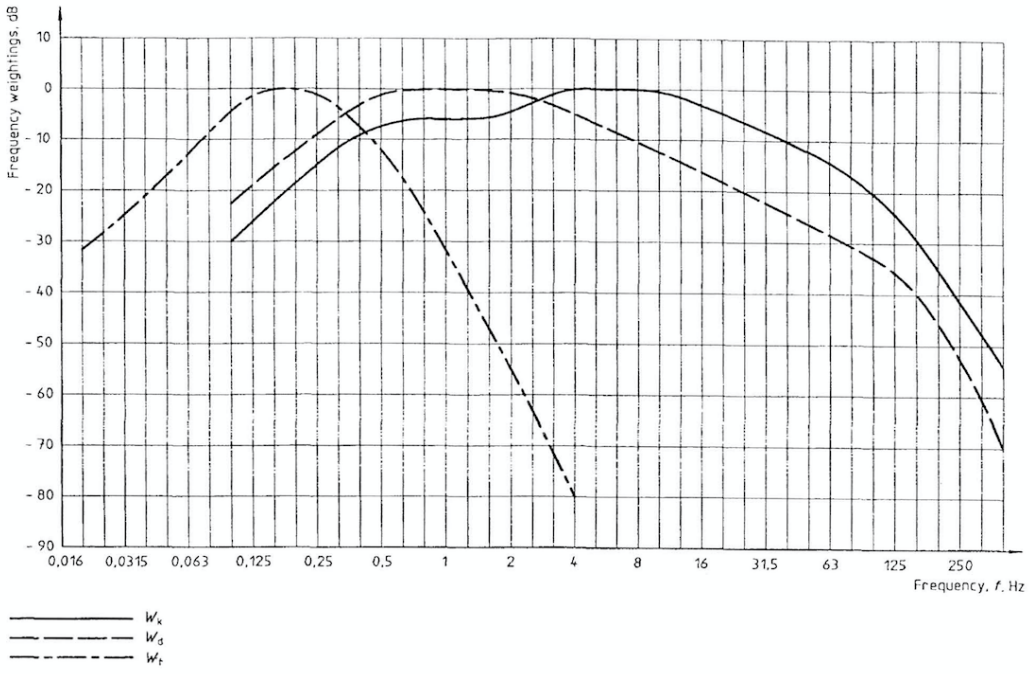


FIGURE 22
Frequency weighting curves for principle weightings [14]

Data Set 08 captured a wider band of the frequency range and therefore will be used to assess health and comfort. The values will still be slightly greater in magnitude than a fully weighted analysis, therefore it should be interpreted as an upper limit or extreme case scenario.

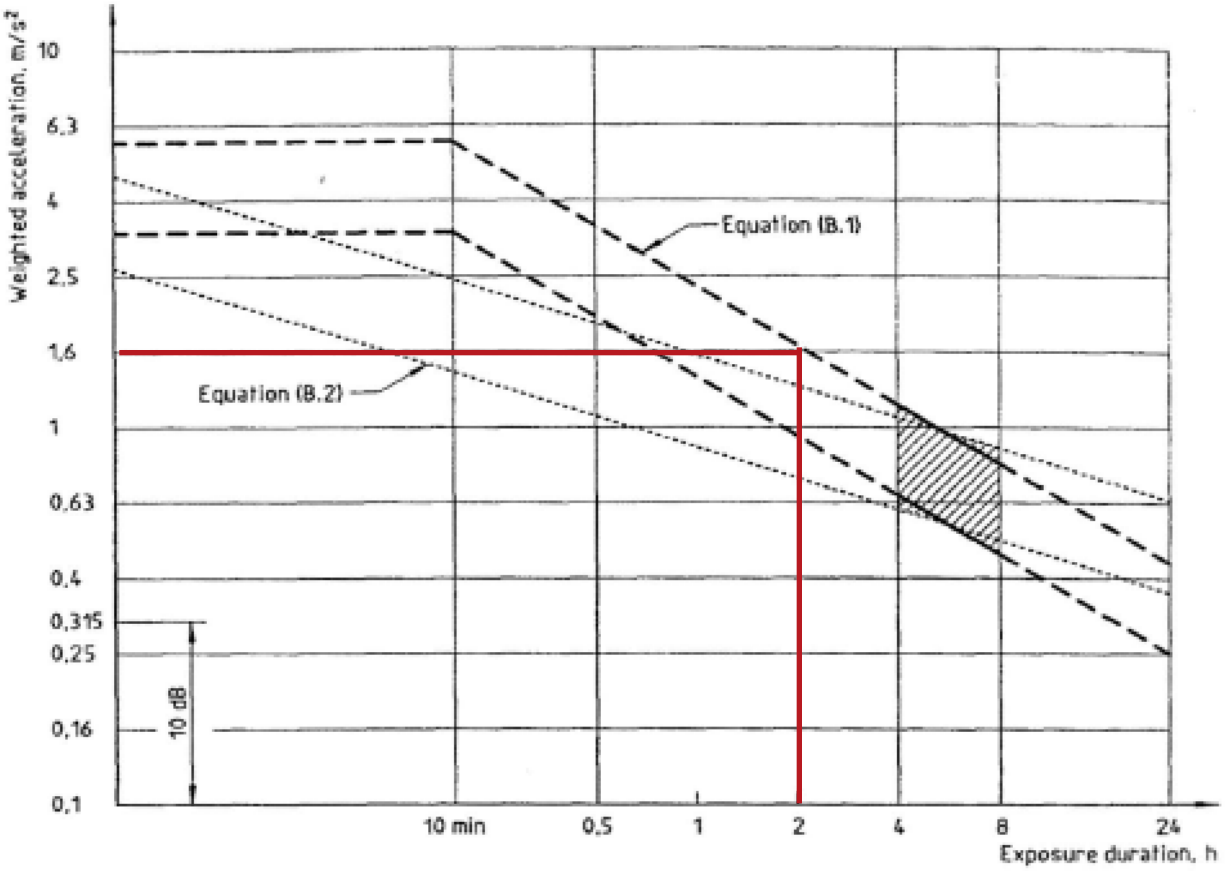


FIGURE 23
Health guidance caution zones for WBV [14]

TABLE 11
Combined RMS Values and health for Data Set 08

	Chassis Frame	Back Plate	Seat Pan
Weighted RMS_{xyz} Acceleration [m/sec^2] (Scaled)	0.6857	0.5736	0.5810
Number of repeated exposures in 2hrs without Health Concerns (Scaled)	2.33	2.79	2.75
Weighted RMS_{xyz} Acceleration [m/sec^2]	0.5914	0.8065	0.4881
Number of repeated exposures in 2hrs without Health Concerns (Unscaled)	2.71	1.98	3.28

Figure 23 shows the health guidance caution zones indicated by ISO 2361-1 with an indication for a two hour exposure time and the corresponding RMS acceleration limit. Base on this chart, if a user were to experience the events and levels of WBV produced during testing, he or she would be able to experience it 2.33 times within two hours before being concerned for his or her health. This is calculated with the application of scaling factors to the axes. These scaling factors are based on relative importance of each axis based on studies done to create the ISO 2631-1 standard. These were conducted for traditional WBV applications and it has been discussed that PWC respond uniquely to vibration inputs. Therefore, based on the data collected for this study, and the discussion regarding the inverted pendulum characteristics, the applicability of these scaling factors is doubted. Both scaled and unscaled analyses are included in Table 11. Without scaling each axis, a person using a PWC would be able to experience it 1.98 times within two hours before being concerned for his or her health.

Another consideration is that perhaps the research team may obtain different results using human operators rather than an ATD and it will be interesting to compare the results of clinicians

to persons who regularly use a PWC. Differences between the ATD and WTD can be seen when comparing VDV magnitudes for Data Set 04 and Data Set 08 where the former was ran using the ATD and measured larger accelerations at the back support. As seen in Table 5 and Table 8, Data Set 04 measured a combined VDV at the back plate of 20.358 VDV while Data Set 04 measured 7.674 VDV. The difference in magnitude cannot solely be due to the differences in frequency weightings because the unweighted VDVs follow the same trend. It is hypothesized that the ATD has a higher center of gravity in its torso considering the fact that it possessed a head, neck, and arms while the WTD was a solid box and did not extend past the top edge of the back support. Also note that both the ATD and WTD were strapped to the back support which would amplify any momentums that the accelerometers would measure. As mentioned previously, it will be interesting to study how the data varies for persons who do and do not use a PWC regularly based on their level of trunk control and how much the back plate holds their mass. The back support may prove to be a critical location for users who have a preexisting spinal condition or lack full control of his or her torso. These considerations should be evaluated in future phase of this study.

Regarding the instrumentation, future steps should be taken to adjust the pwcspSDSensorNode.ino sketch to control and obtain sampling rates of at least 160 Hz. Also, there is some promise in the XBee and Processing integration. The researcher had no prior experience with wireless communication nor programming using processing before this study therefore with additional attention and expertise, one might be able to trouble shoot the issues and make this a truly wireless system. That could one day lead to being able to monitor a user remotely to track vibration exposure levels or allowing clinicians to send this system home with

a user and collect data over the course of several hours or days. Real world data would allow one to better understand the magnitudes and dosages a user experiences.

Chapter 5: Conclusion

The purpose of this study was to develop an adaptable system to measure acceleration data from multiple points of interest (POIs) on power wheelchairs simultaneously. Also, its purpose was to determine the transmissibility of whole body vibration at these points of interest. The first goal was met with the development of a nodal sensor system which utilized the Arduino micro controller platform. The system integrates an accelerometer with an Arduino Uno, a SD data logger to collect and temporarily store the data. Development work was conducted to stream data wirelessly over XBee RF radios to a base computer running a Processing script that would produce a realtime plot however due to time constraints was halted for this study. The system provides a simple, modular solution for collecting motion data from multiple locations and can be utilized for future mobility studies.

An anthropometric test device and wheelchair test device were used while the chair traversed a standardized road course. Acceleration data was collected at the frame, seat pan, and back support to determine transmissibility. Vibration magnitudes were evaluated using vibration dose values and supplemented by root-mean-squared accelerations. Seat effective amplitude transmissibility was calculated from the chassis frame to both the back plate and seat pan and it was determined that vibration magnitudes were increased at the back plate along the x-axis more than what is typically expected or considered for traditional whole body vibration analyses. This is primarily due to the unique characteristics of the mid-wheel drive power wheelchair.

5.1 Contributions

This study has provided a greater understanding of the vibrational characteristics of power wheelchairs. It has shed light on the differences between traditional WBV studies and the unique implications for users of these devices. Methodology and instrumentation has been laid out to continue work in this area and the development of the sensor node measurement system can be utilized for the remainder of the larger multi-phase study. It will be instrumental in testing multiple models of power wheelchairs and providing useful information for researchers, clinicians, and manufacturers of power wheelchairs.

5.2 Additional Applications

One foreseeable application outside of the scope of this study include Clinical/ in-home testing using XBee data transmission. Upon successful follow through with the work started here, the integration of XBee wireless transmission with a realtime display of the data would be useful in testing devices around a large clinic. It would also open possibilities for remote testing and data collection in a person's home or in the real world where it is possible to gather data that would be applicable to a large number of users.

This system can also be used as a tool for assessing which out of several devices would be best for a particular user. Based on a user's unique medical history and needs, clinicians could repeat a portion of this study to determine if a particular device would pose a health concern over an extended period of time. This could prevent or diagnose injuries and discomfort sooner.

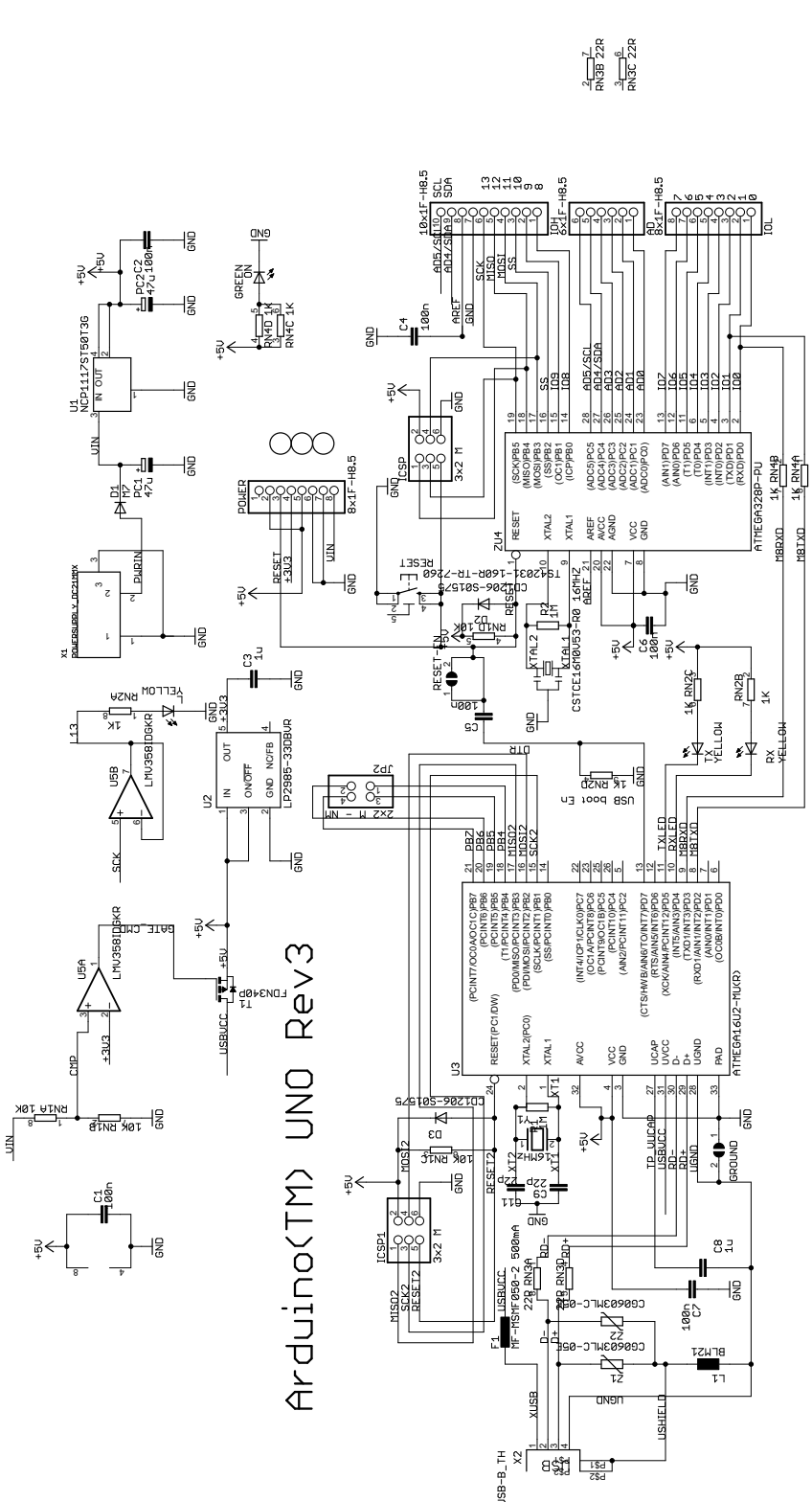
Finally, the XBee and arduino could be applied to any number of remote sensing applications and if made open source would be useful to numerous researchers and hobbyists alike who could continue, modify, and improve upon its development.

5.3 Future Work

Future work will include testing multiple PWCs, including Invacare's prototype, to compare their performance in reducing WBV at the seat pan and back plate. Human trials will then be conducted with both clinicians and users of PWC. Based on these trial, findings will be combined with focus group data collected from users and clinicians in a prior phase. All of this information will be synthesized and conclusions will be reported.

APPENDIX A

Arduino Uno Rev 3 Pin Out Diagram [10]



APPENDIX B

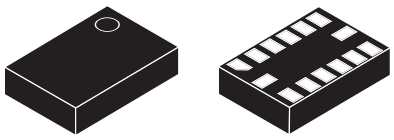
LSM303DLHC Data Sheet (full document available at www.st.com)

life.augmented

LSM303DLHC

Ultra-compact high-performance eCompass module: 3D accelerometer and 3D magnetometer

Datasheet - production data



LGA-14 (3x5x1mm)

Features

- 3 magnetic field channels and 3 acceleration channels
- From ±1.3 to ±8.1 gauss magnetic field full scale
- ±2g/±4g/±8g/±16g linear acceleration full scale
- 16-bit data output
- I^2C serial interface
- Analog supply voltage 2.16 V to 3.6 V
- Power-down mode / low-power mode
- 2 independent programmable interrupt generators for free-fall and motion detection
- Embedded temperature sensor
- Embedded FIFO
- 6D/4D-orientation detection
- ECOPACK® RoHS and “Green” compliant

Applications

- Tilt-compensated compasses
- Map rotation
- Position detection
- Motion-activated functions
- Free-fall detection
- Click/double-click recognition
- Pedometers
- Intelligent power-saving for handheld devices
- Display orientation
- Gaming and virtual reality input devices
- Impact recognition and logging
- Vibration monitoring and compensation

Description

The LSM303DLHC is a system-in-package featuring a 3D digital linear acceleration sensor and a 3D digital magnetic sensor.

The LSM303DLHC has linear acceleration full scales of ±2*g* / ±4*g* / ±8*g* / ±16*g* and a magnetic field full scale of ±1.3 / ±1.9 / ±2.5 / ±4.0 / ±4.7 / ±5.6 / ±8.1 gauss.

The LSM303DLHC includes an I^2C serial bus interface that supports standard and fast mode 100 kHz and 400 kHz. The system can be configured to generate interrupt signals by inertial wake-up/free-fall events as well as by the position of the device itself. Thresholds and timing of interrupt generators are programmable by the end user. Magnetic and accelerometer blocks can be enabled or put into power-down mode separately.

The LSM303DLHC is available in a plastic land grid array package (LGA) and is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

Table 1. Device summary

Part number	Temperature range [°C]	Package	Packing
LSM303DLHC	-40 to +85	LGA-14	Tray
LSM303DLHCTR	-40 to +85	LGA-14	Tape and reel

APPENDIX C

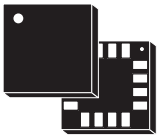
L3GD20 Data Sheet (full document available at www.st.com)

life.augmented

L3GD20

MEMS motion sensor:
three-axis digital output gyroscope

Datasheet - production data



LGA-16 (4x4x1 mm)

Features

- Three selectable full scales (250/500/2000 dps)
- I^2C/SPI digital output interface
- 16 bit-rate value data output
- 8-bit temperature data output
- Two digital output lines (interrupt and data ready)
- Integrated low- and high-pass filters with user-selectable bandwidth
- Wide supply voltage: 2.4 V to 3.6 V
- Low voltage-compatible IOs (1.8 V)
- Embedded power-down and sleep mode
- Embedded temperature sensor
- Embedded FIFO
- High shock survivability
- Extended operating temperature range (-40 °C to +85 °C)
- ECOPACK® RoHS and “Green” compliant

Applications

- Gaming and virtual reality input devices
- Motion control with MMI (man-machine interface)
- GPS navigation systems
- Appliances and robotics

Description

The L3GD20 is a low-power three-axis angular rate sensor.

It includes a sensing element and an IC interface capable of providing the measured angular rate to the external world through a digital interface (I^2C/SPI).

The sensing element is manufactured using a dedicated micro-machining process developed by STMicroelectronics to produce inertial sensors and actuators on silicon wafers.

The IC interface is manufactured using a CMOS process that allows a high level of integration to design a dedicated circuit which is trimmed to better match the sensing element characteristics. The L3GD20 has a full scale of ±250/±500/ ±2000 dps and is capable of measuring rates with a user-selectable bandwidth.

The L3GD20 is available in a plastic land grid array (LGA) package and can operate within a temperature range of -40 °C to +85 °C.

Table 1. Device summary

Order code	Temperature range (°C)	Package	Packing
L3GD20	-40 to +85	LGA-16 (4x4x1 mm)	Tray
L3GD20TR	-40 to +85	LGA-16 (4x4x1 mm)	Tape and reel

APPENDIX D

dof_9.ino Arduino Sketch

```
/*  dof_9  ver 1.0  July 28, 2014

    by L'Nard Tufts, The Ohio State University

    adapted from pitchrollheading by Adafruit Industries

    This program utilizes the Adafruit 9 DOF breakout board
    to output the following measurements to the serial monitor:
        Acceleration (X, Y, and Z axes) in m/s^2
        Orientation (Pitch, Roll, Heading) in degrees
*/

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_9DOF.h>

/* Assign a unique ID to the sensors */
Adafruit_9DOF                dof   = Adafruit_9DOF();
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   mag   = Adafruit_LSM303_Mag_Unified(30302);

/* Update this with the correct SLP for accurate altitude measurements */
float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;

/**************************************************************************/
/*!
    @brief  Initialises all the sensors used by this example
*/
/**************************************************************************/
void initSensors()
{
  if(!accel.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println(F("Ooops, no LSM303 accelerometer detected ... Check your wiring!"));
    while(1);
  }
  if(!mag.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println("Ooops, no LSM303 magnetometer detected ... Check your wiring!");
    while(1);
  }
}

void displaySensorDetails(void)
{
  sensor_t sensor;
  accel.getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
```

```
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" m/s^2
  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println(" m/s^2
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" m/s^
  Serial.println("------------------------------------");
  Serial.println("");
  delay(500);
}
/**************************************************************************/

/**************************************************************************/
void setup(void)
{
  Serial.begin(115200);
  Serial.println(F("Adafruit 9 DOF Pitch/Roll/Heading Example")); Serial.println("");

  /* Initialise the sensors */
  initSensors();

  /* Display some basic information on this sensor */
  displaySensorDetails();
}

/**************************************************************************/
/*!
    @brief  Constantly check the acceleration/roll/pitch/heading
*/
/**************************************************************************/
void loop(void)
{
  sensors_event_t accel_event;
  sensors_event_t mag_event;
  sensors_vec_t   orientation;

  /* Display the results (acceleration is measured in m/s^2) */
  Serial.print(F("X: "));
  Serial.print(accel_event.acceleration.x);
  Serial.print(F("; "));
  Serial.print(F("Y: "));
  Serial.print(accel_event.acceleration.y);
  Serial.print(F("; "));
  Serial.print(F("Z: "));
  Serial.print(accel_event.acceleration.z);
  Serial.print(F("; "));

  /* Calculate pitch and roll from the raw accelerometer data */
  accel.getEvent(&accel_event);
  if (dof.accelGetOrientation(&accel_event, &orientation))
  {
    /* 'orientation' should have valid .roll and .pitch fields */
    Serial.print(F("Roll: "));
```

```
    Serial.print(orientation.roll);
    Serial.print(F("; "));
    Serial.print(F("Pitch: "));
    Serial.print(orientation.pitch);
    Serial.print(F("; "));
  }

  /* Calculate the heading using the magnetometer */
  mag.getEvent(&mag_event);
  if (dof.magGetOrientation(SENSOR_AXIS_Z, &mag_event, &orientation))
  {
    /* 'orientation' should have valid .heading data now */
    Serial.print(F("Heading: "));
    Serial.print(orientation.heading);
    Serial.print(F("; "));
  }


  delay(1/25600);
}
```

APPENDIX E

Xively_9dof_cc3300_v1.ino Arduino Sketch

```
/*CC3300 & Xively*/

/*************************************************
  This is a sketch to use the CC3000 WiFi chip & Xively

  Written by Marco Schwartz for Open Home Automation
 *************************************************/

// Libraries
#include <Adafruit_CC3000.h>
#include <SPI.h>
#include <Wire.h>
#include <avr/wdt.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_9DOF.h>

// Define CC3000 chip pins
#define ADAFRUIT_CC3000_IRQ   3
#define ADAFRUIT_CC3000_VBAT  5
#define ADAFRUIT_CC3000_CS    10

/* Assign a unique ID to the sensors */
Adafruit_9DOF                dof   = Adafruit_9DOF();
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   mag   = Adafruit_LSM303_Mag_Unified(30302);

/* Update this with the correct SLP for accurate altitude measurements */
float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;

// Create CC3000 instances
Adafruit_CC3000 cc3000 = Adafruit_CC3000(ADAFRUIT_CC3000_CS, ADAFRUIT_CC3000_IRQ, ADAFRUIT
                                         SPI_CLOCK_DIV2); // you can change this clock spe


// WLAN parameters
#define        WLAN_SSID"AndroidAP"
#define        WLAN_PASS"fgabbcdef"
#define WLAN_SECURITY   WLAN_SEC_WPA2

/* The next set of parameters concerns Xively. You need the following
 parameters of your Xively account: your API key and your feedID.
 These will be used to make the request to the Xively server. */

// Xively parameters
#define   API_key"zSfnbS9wSEnUJ3IiTEM31PQ4KkcCQw2Hgk2wExRQfKQYxzgM"
#define   feedID"588416479"
int buffer_size = 50;

uint32_t ip;

void initSensors()
```

```
{
  if(!accel.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println(F("Ooops, no LSM303 accelerometer detected ... Check your wiring!"));
    while(1);
  }
  if(!mag.begin())
  {
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println("Ooops, no LSM303 magnetometer detected ... Check your wiring!");
    while(1);
  }
}

/* Now, we enter the setup() part of the sketch. As we will connect
 and disconnect from the Xively server every time we want to send
 data, the setup() we will only include the initialization of the
 CC3000 chip, and the connection to the WiFi network: */

void setup(void)
{
  // Initialize
  Serial.begin(115200);

  /* Initialise the sensors */
  initSensors();

  Serial.println(F("Initializing WiFi chip..."));
  if (!cc3000.begin())
  {
    Serial.println(F("Couldn't begin()! Check your wiring?"));
    while(1);
  }

  // Connect to WiFi network
  Serial.print(F("Connecting to WiFi network ..."));
  cc3000.connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY);
  Serial.println(F("done!"));

  // Wait for DHCP to complete
  Serial.println(F("Request DHCP"));
  while (!cc3000.checkDHCP())
  {
    delay(100);
  }

}

void loop(void)
{
    // Start watchdog
```

```
 wdt_enable(WDTO_8S);

// Get IP
uint32_t ip = 0;
Serial.print(F("api.xively.com -> "));
while  (ip  ==  0)  {
  if  (!  cc3000.getHostByName("api.xively.com", &ip))  {
    Serial.println(F("Couldn't resolve!"));
    while(1){}
  }
  delay(500);
  }
cc3000.printIPdotsRev(ip);
Serial.println(F(""));

// Get data & transform to integers
sensors_event_t accel_event;
sensors_event_t mag_event;
sensors_vec_t   orientation;

float a_x = accel_event.acceleration.x;
float a_y = accel_event.acceleration.y;
float a_z = accel_event.acceleration.z;
Serial.print(a_x);
Serial.print(a_y);
Serial.print(a_z);

/* Calculate pitch and roll from the raw accelerometer data */
//accel.getEvent(&accel_event);
//if (dof.accelGetOrientation(&accel_event, &orientation))
//{
  /* 'orientation' should have valid .roll and .pitch fields */
 // float roll = orientation.roll;
 // float pitch = orientation.pitch;
//}

/* Calculate the heading using the magnetometer */
//mag.getEvent(&mag_event);
//if (dof.magGetOrientation(SENSOR_AXIS_Z, &mag_event, &orientation))
//{
  /* 'orientation' should have valid .heading data now */
//  float head = orientation.heading;
// }

//Skipped interger conversion. Fix if necessary

int AccelerationX = (int) a_x;
int AccelerationY = (int) a_y;
int AccelerationZ = (int) a_z;
//int Roll = (int) orientation.roll;
//int Pitch = (int) orientation.pitch;
//int Heading = (int) orientation.heading;
```

```
// Prepare JSON for Xively & get length
int length = 0;

// JSON data
String data ="";
data = data +"\n"  +"{\"version\":\"1.0.0\",\"datastreams\" : [ "
 +"{\"id\" : \"AccelerationX\",\"current_value\" : \""  +String(AccelerationX)  +"\"},"
 +"{\"id\" : \"AccelerationY\",\"current_value\" : \""  +String(AccelerationY)  +"\"},"
 +"{\"id\" : \"AccelerationZ\",\"current_value\" : \""  +String(AccelerationZ)  +"\"}]}"

// Get length
length = data.length();

// Reset watchdog
wdt_reset();

// Check connection to WiFi
Serial.print(F("Checking WiFi connection ..."));
if(!cc3000.checkConnected()){while(1){}}
Serial.println(F("done."));
wdt_reset();

// Ping Xively server
 Serial.print(F("Pinging Xively server ..."));
if(!cc3000.ping(ip, 2)){while(1){}}
Serial.println(F("done."));
wdt_reset();

// Send request
Adafruit_CC3000_Client client = cc3000.connectTCP(ip, 80);
if (client.connected()) {
  Serial.println(F("Connected to Xively server."));

  // Send headers
  Serial.print(F("Sending headers "));
  client.fastrprint(F("PUT /v2/feeds/"));
  client.fastrprint(feedID);
  client.fastrprintln(F(".json HTTP/1.0"));
  Serial.print(F("."));
  client.fastrprintln(F("Host: api.xively.com"));
  Serial.print(F("."));
  client.fastrprint(F("X-ApiKey: "));
  client.fastrprintln(API_key);
  Serial.print(F("."));
  client.fastrprint(F("Content-Length: "));
  client.println(length);
  Serial.print(F("."));
  client.fastrprint(F("Connection: close"));
  Serial.println(F(" done."));
  // Reset watchdog
  wdt_reset();
```

```
    // Send data
    Serial.print(F("Sending data ..."));
    client.fastrprintln(F(""));
      sendData(client,data,buffer_size);
    client.fastrprintln(F(""));
    Serial.println(F("done."));

    // Reset watchdog
    wdt_reset();

   }else {
    Serial.println(F("Connection failed"));
    return;
  }

  // Reset watchdog
  wdt_reset();

  Serial.println(F("Reading answer ..."));
  while (client.connected()) {
    while (client.available()) {
      char c = client.read();
      Serial.print(c);
    }
  }

  // Reset watchdog
  wdt_reset();

  // Close connection and disconnect
  client.close();
  Serial.println(F("Closing connection"));

  // Reset watchdog & disable
  wdt_reset();
  wdt_disable();

  // Wait 10 seconds until next update
  wait(10000);

}

// Send data chunk by chunk
void  sendData(Adafruit_CC3000_Client&  client,String  input,int chunkSize) {

  // Get String length
  int length = input.length();
  int max_iteration = (int)(length/chunkSize);

  for (int i = 0; i < length; i++) {
    client.print(input.substring(i*chunkSize, (i+1)*chunkSize));
```

```
    wdt_reset();
    }
}

// Wait for a given time using the watchdog
void wait(int total_delay) {

  int number_steps = (int)(total_delay/5000);
  wdt_enable(WDTO_8S);
  for (int i = 0; i < number_steps; i++){
    delay(5000);
    wdt_reset();
  }
  wdt_disable();
}
```

APPENDIX F

ADXL335 Data Sheet (full document available at www.analog.com)

ANALOG DEVICES

Small, Low Power, 3-Axis ±3 *g* Accelerometer

ADXL335

FEATURES

3-axis sensing
Small, low profile package
4 mm × 4 mm × 1.45 mm LFCSP
Low power : 350 µA (typical)
Single-supply operation: 1.8 V to 3.6 V
10,000 *g* shock survival
Excellent temperature stability
BW adjustment with a single capacitor per axis
RoHS/WEEE lead-free compliant

APPLICATIONS

Cost sensitive, low power, motion- and tilt-sensing applications
Mobile devices
Gaming systems
Disk drive protection
Image stabilization
Sports and health devices

GENERAL DESCRIPTION

The ADXL335 is a small, thin, low power, complete 3-axis accelerometer with signal conditioned voltage outputs. The product measures acceleration with a minimum full-scale range of ±3 *g*. It can measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion, shock, or vibration.

The user selects the bandwidth of the accelerometer using the C_X, C_Y, and C_Z capacitors at the X_{OUT}, Y_{OUT}, and Z_{OUT} pins. Bandwidths can be selected to suit the application, with a range of 0.5 Hz to 1600 Hz for the X and Y axes, and a range of 0.5 Hz to 550 Hz for the Z axis.

The ADXL335 is available in a small, low profile, 4 mm × 4 mm × 1.45 mm, 16-lead, plastic lead frame chip scale package (LFCSP_LQ).

FUNCTIONAL BLOCK DIAGRAM

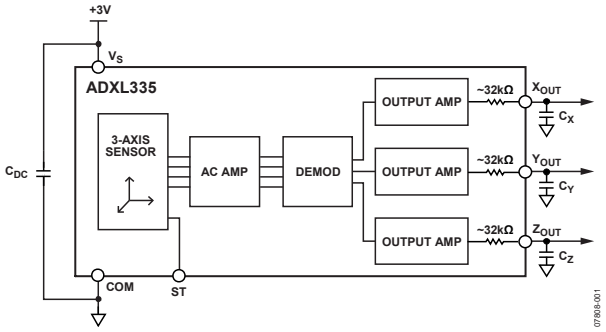


Figure 1.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 www.analog.com
Fax: 781.461.3113

APPENDIX G

XBee Coordinator Configuration Profile

```
<?xml version="1.0" encoding="UTF-8"?>

<data>
  <profile>
    <description_file>XBP24-ZB_21A7_S2B.xml</description_file>
    <settings>
      <setting command="ID">2001</setting>
      <setting command="SC">7FFF</setting>
      <setting command="SD">3</setting>
      <setting command="ZS">0</setting>
      <setting command="NJ">FF</setting>
      <setting command="DH">0</setting>
      <setting command="DL">0</setting>
      <setting command="NI">0x20</setting>
      <setting command="NH">30</setting>
      <setting command="BH">0</setting>
      <setting command="AR">FF</setting>
      <setting command="DD">30000</setting>
      <setting command="NT">3C</setting>
      <setting command="NO">0</setting>
      <setting command="CR">3</setting>
      <setting command="PL">4</setting>
      <setting command="PM">1</setting>
      <setting command="EE">0</setting>
      <setting command="EO">0</setting>
      <setting command="KY"></setting>
      <setting command="NK"></setting>
      <setting command="BD">7</setting>
      <setting command="NB">0</setting>
      <setting command="SB">1</setting>
      <setting command="D7">1</setting>
      <setting command="D6">0</setting>
      <setting command="AP">2</setting>
      <setting command="AO">0</setting>
      <setting command="SP">20</setting>
      <setting command="SN">1</setting>
      <setting command="D0">0</setting>
      <setting command="D1">0</setting>
      <setting command="D2">0</setting>
      <setting command="D3">0</setting>
      <setting command="D4">0</setting>
      <setting command="D5">1</setting>
      <setting command="P0">1</setting>
      <setting command="P1">0</setting>
      <setting command="P2">0</setting>
      <setting command="PR">1FFF</setting>
      <setting command="LT">0</setting>
      <setting command="RP">28</setting>
      <setting command="DO">1</setting>
      <setting command="IR">32</setting>
      <setting command="IC">0</setting>
      <setting command="V+">0</setting>
    </settings>
  </profile>
</data>
```

APPENDIX H

XBee Router Configuration Profile

```
<?xml version="1.0" encoding="UTF-8"?>

<data>
  <profile>
    <description_file>XBP24-ZB_22A7_S2B.xml</description_file>
    <settings>
      <setting command="ID">2001</setting>
      <setting command="SC">7FFF</setting>
      <setting command="SD">3</setting>
      <setting command="ZS">0</setting>
      <setting command="NJ">FF</setting>
      <setting command="NW">0</setting>
      <setting command="JV">1</setting>
      <setting command="JN">0</setting>
      <setting command="DH">0</setting>
      <setting command="DL">0</setting>
      <setting command="NI">0x20</setting>
      <setting command="NH">30</setting>
      <setting command="BH">0</setting>
      <setting command="AR">FF</setting>
      <setting command="DD">30000</setting>
      <setting command="NT">3C</setting>
      <setting command="NO">0</setting>
      <setting command="CR">3</setting>
      <setting command="SE">E8</setting>
      <setting command="DE">E8</setting>
      <setting command="CI">11</setting>
      <setting command="PL">4</setting>
      <setting command="PM">1</setting>
      <setting command="EE">0</setting>
      <setting command="EO">0</setting>
      <setting command="KY"></setting>
      <setting command="BD">7</setting>
      <setting command="NB">0</setting>
      <setting command="SB">1</setting>
      <setting command="RO">3</setting>
      <setting command="D7">1</setting>
      <setting command="D6">0</setting>
      <setting command="CT">64</setting>
      <setting command="GT">3E8</setting>
      <setting command="CC">2B</setting>
      <setting command="SM">0</setting>
      <setting command="SN">1</setting>
      <setting command="SO">0</setting>
      <setting command="SP">20</setting>
      <setting command="ST">1388</setting>
      <setting command="PO">0</setting>
      <setting command="D0">2</setting>
      <setting command="D1">2</setting>
      <setting command="D2">2</setting>
      <setting command="D3">0</setting>
      <setting command="D4">0</setting>
      <setting command="D5">1</setting>
      <setting command="P0">1</setting>
      <setting command="P1">0</setting>
      <setting command="P2">0</setting>
      <setting command="PR">1FFF</setting>
      <setting command="LT">0</setting>
      <setting command="RP">28</setting>
```

```
      <setting command="DO">1</setting>
      <setting command="IR">32</setting>
      <setting command="IC">0</setting>
      <setting command="V+">0</setting>
    </settings>
  </profile>
</data>
```

APPENDIX I

pwcspZBSensorNode.ino Sensor Module Arduino Sketch

```
/*
 Transmit Tri-Axial Accelerometer Readings to a Base Coordinator XBees

 For use in point-to-point or multipoint-to-point wireless sensor
 network. Used to take tri-axial accelerometer readings, save data to
 SD card, and transmit readings to a coordinator XBee. Can also be used
 to calibrate an Adafruit ADXL335, ADXL326, or ADXL377 Analog
 Accelerometer Breakout Board and print out readings to the serial
 monitor.

 Expected Setup:

 [XBee Pro - Series 2]
 | | |
 V V V
 [Sparkfun XBee Shield]
 | | |
 V V V
 [Adafruit Data Logging Shield]
 | | |
 V V V
 [Arduino Uno]

 Used with XBee radio configured as follows:
 ZigBee Router AT
 BD - 115200 [7]
 NB - N [0]
 SB - 2 [1]

 Full profile can be viewed at:
 smb://shrs-p01/users/pwcsp/Development/Configuration/profile_xb_router.xml

 Last modified January 7, 2015
 by L'Nard E.T. Tufts II | Student Research Assistant
 */


/*Declare Required Libraries for Sketch*/
#include <SD.h>
#include <Wire.h>
#include "RTClib.h"

/*Declare Variable Names for Pins Used in Sketch*/
const int xInput = A0; //Analog Input Pin 0
const int yInput = A1; //Analog Input Pin 1
const int zInput = A2; //Analog Input Pin 2
const int buttonPin = 2; //Used for calibration

int xSend = 3; //Digital I/O (PWM) Pin 3
int ySend = 6; //Digital I/O (PWM) Pin 6
int zSend = 5; //Digital I/O (PWM) Pin 5

/*****************************************************************/
/****CHANGE BASED ON WHICH SENSOR NODE THIS SKETCH IS UPLOADED TO****/
// Arm Rest:   DD = 22
// Chasis:     B1 = 23
// Foot Plate: 9F = 24

int sensortag = 22;
/*****************************************************************/
```

```
/* Calibration */
/*
 Raw Ranges:
 If unknown, initialize to mid-range (512) and allow calibration to find
 the minimum and maximum raw values for each axis. Once determined, set
 to calibrated values.
 */
int xRawMin = 406; // 3G: 406  5G: 490
int xRawMax = 608; // 3G: 608  5G: 530

int yRawMin = 400; // 3G: 490  5G: 400
int yRawMax = 600; // 3G: 530  5G: 600

int zRawMin = 422; // 3G: 422  5G: 495
int zRawMax = 627; // 3G: 627  5G: 535

/* End Calibration */

/* SD Card Logging */
// take multiple samples to reduce noise
const int sampleSize = 10;

// how many milliseconds between entries; grabbing data and logging it
#define LOG_INTERVAL  10 // (reduce to take more/faster data)

/*
how many milliseconds before writing the logged data permanently to disk
 set it to the LOG_INTERVAL to write each time (safest)
 set it to 10*LOG_INTERVAL to write all data every 10 datareads
 you could lose up to the last 10 reads if power is lost but it uses less
 power and is much faster! */
#define SYNC_INTERVAL 10 // mills between calls to flush() to write data to card
uint32_t syncTime = 0; // time of last sync()

/****SWITCH ON (1) OR OFF (0) BASED ON DESIRED OUTPUTS****/
#define ECHO_TO_SERIAL   1 // echo data to serial port
#define WAIT_TO_START    0 // wait for serial input in setup()
#define SEND_TO_XBEE     0 // in UART Mode, write to serial to send over XBee
#define WRITE_TO_PIN     1 // write to digital output pins if directly wired to XBee

RTC_DS1307 RTC; // define the Real Time Clock object

// for the data logging shield, we use digital pin 10 for the SD cs line
const int chipSelect = 10;

// the logging file
File logfile;

/*This function returns the date and time for SD card file access and
 modify time. One needs to call in setup() to register this callback
 function: SdFile::dateTimeCallback(file_date_time);*/
void dateTime(uint16_t* date, uint16_t* time)
{
  DateTime now = RTC.now();
  *date=FAT_DATE(now.year(),now.month(),now.day());
  *time=FAT_TIME(now.hour(),now.minute(),now.second());
}
/* End SD Card Logging */

/*This function prints an error message and stops the sketch in the
 event of an  error */
void error(char *str)
{
```

```
#if ECHO_TO_SERIAL
  Serial.print("error: ");
  Serial.println(str);
#endif
  while(1);
}


/*Setup function runs once to initialize pin modes, the serial port, and
 the logging file */
void setup(void){
  analogReference(EXTERNAL); // IMPOTRTANT if AREF is connected to accel
  Serial.begin(115200);
  /*Make sure the Serial.begin(#) baud rate matches the baud rate of all
   XBees, Arduinos, and Processing sketches used in this project*/

  pinMode(xSend, OUTPUT);
  pinMode(ySend, OUTPUT);
  pinMode(zSend, OUTPUT);

#if WAIT_TO_START
  Serial.println("Type any character to start");
  while (!Serial.available());
#endif //WAIT_TO_START

  /*SD Card Setup*/
  // make sure that the default chip select pin is set to output, even
  // if you don't use it:
  pinMode(10, OUTPUT);

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    error("Card failed, or not present");
  }
#if ECHO_TO_SERIAL
  Serial.println("card initialized.");
#endif

  // Attach callback function to provide date time to files
  //SdFile::dateTimeCallback(dateTime);

  // create a new file
  char filename[] = "LOGGER00.CSV"; //Optional: change to desired filename
  for (uint8_t i = 0; i < 100; i++) {
    // Change the counter for the file
    filename[6] = i/10 + '0'; // Tens place
    filename[7] = i%10 + '0'; // Ones place
    if (! SD.exists(filename)) {
      // only open a new file if it doesn't exist
      logfile = SD.open(filename, FILE_WRITE);
      break;  // leave the loop!
    }
  }

  if (! logfile) {
    error("couldnt create file");
  }

#if ECHO_TO_SERIAL
  Serial.print("Logging to: ");
  Serial.println(filename);
#endif //ECHO_TO_SERIAL

  // connect to the Real Time Clock (RTC)
```

```
  Wire.begin();
  if (!RTC.begin()) {
    logfile.println("RTC failed");
#if ECHO_TO_SERIAL
    Serial.println("RTC failed");
#endif  //ECHO_TO_SERIAL
  }

  // print header row to file
  logfile.println("millis,x_min,x_max,y_min,y_max,z_min,z_max,x_raw,y_raw,z_raw,x_scal
ed,y_scaled,z_scaled,x_accel,y_accel,z_accel");
#if ECHO_TO_SERIAL
  Serial.println("millis,a_x,a_y,a_z");
#endif //ECHO_TO_SERIAL
}


/
*Loop function repeats consecutively until power is removed or Arduino board is reset.
 Reads, transmits, and logs accelerometer data. Also contains code for calibrating the

 sensor.
 */
void loop (){
  // delay for the amount of time we want between readings
  delay((LOG_INTERVAL - 1) - (millis() % LOG_INTERVAL));

  /* READ DATA */
  //Call ReadAxis() function to read pins (See Function after loop ())
  int xRaw = ReadAxis(xInput);
  int yRaw = ReadAxis(yInput);
  int zRaw = ReadAxis(zInput);
  uint32_t m = millis(); // note time data was taken (milliseconds since start)
  /* END READ DATA */

  /* TRANSMIT DATA */
#if WRITE_TO_PIN
  analogWrite(xSend,xRaw/4); // Divide by 4 to convert 10-bit (read) to 8-bit (write)
  analogWrite(ySend,yRaw/4);
  analogWrite(zSend,zRaw/4);
#endif // WRITE_TO_PIN

#if SEND_TO_XBEE
  Serial.print(xRaw);
  Serial.print(yRaw);
  Serial.print(zRaw);
  Serial.println(sensortag);
#endif // SEND_TO_XBEE
  /* END TRANSMIT DATA */

  /* CALIBRATION */
  // calibrate during button press using AutoCalibrate function
  if (digitalRead(buttonPin) == LOW)
  {
    AutoCalibrate(xRaw, yRaw, zRaw);
  }
  /* END CALIBRATION */

  /* RAW CONVERSION */
  // Convert raw values to 'milli-Gs"
  long xScaled = map(xRaw, xRawMin, xRawMax, -1000, 1000);
  long yScaled = map(yRaw, yRawMin, yRawMax, -1000, 1000);
  long zScaled = map(zRaw, zRawMin, zRawMax, -1000, 1000);
```

```
  // re-scale to m/s^2
  float xAccel = (xScaled / 1000.0)*(-9.81);
  float yAccel = (yScaled / 1000.0)*(-9.81);
  float zAccel = (zScaled / 1000.0)*(-9.81);
  /* END RAW CONVERSION */

  /* DATA LOGGING */
  /* writes values to SD logging file as comma separated values */

  // Read Time

  logfile.print(m);
  logfile.print(",");

  // Calibrated or Defined Raw Ranges

  logfile.print(xRawMin);
  logfile.print(",");
  logfile.print(xRawMax);
  logfile.print(",");

  logfile.print(yRawMin);
  logfile.print(",");
  logfile.print(yRawMax);
  logfile.print(",");

  logfile.print(zRawMin);
  logfile.print(",");
  logfile.print(zRawMax);
  logfile.print(", ");

  // Raw Values

  logfile.print(xRaw);
  logfile.print(", ");
  logfile.print(yRaw);
  logfile.print(", ");
  logfile.print(zRaw);
  logfile.print(", ");

  // Scaled Values

  logfile.print(xScaled);
  logfile.print(", ");
  logfile.print(yScaled);
  logfile.print(", ");
  logfile.print(zScaled);
  logfile.print(", ");

  // Acceleration Values

  logfile.print(xAccel);
  logfile.print(", ");
  logfile.print(yAccel);
  logfile.print(", ");
  logfile.println(zAccel); // End line


#if ECHO_TO_SERIAL // echo log file to serial monitor
  Serial.print(m);
  Serial.print("     ");

  Serial.print("Raw Ranges: X: ");    // Raw Ranges
  Serial.print(xRawMin);
```

```
  Serial.print("-");
  Serial.print(xRawMax);

  Serial.print(", Y: ");
  Serial.print(yRawMin);
  Serial.print("-");
  Serial.print(yRawMax);

  Serial.print(", Z: ");
  Serial.print(zRawMin);
  Serial.print("-");
  Serial.print(zRawMax);

  Serial.print("     ");     //Raw Values

  Serial.print(xRaw);
  Serial.print(", ");
  Serial.print(yRaw);
  Serial.print(", ");
  Serial.print(zRaw);

  Serial.print("     ");    //Scaled Values

  Serial.print(xScaled);
  Serial.print(", ");
  Serial.print(yScaled);
  Serial.print(", ");
  Serial.print(zScaled);

  Serial.print("     ");    //Acceleration Values

  Serial.print(xAccel);
  Serial.print(", ");
  Serial.print(yAccel);
  Serial.print(", ");
  Serial.println(zAccel);
#endif // ECHO_TO_SERIAL

  // Now we write data to disk! Don't sync too often - requires 2048 bytes of I/O to
SD card
  // which uses a bunch of power and takes time
  if ((millis() - syncTime) < SYNC_INTERVAL) return;
  syncTime = millis();

  logfile.flush();

  /* END DATA LOGGING */
} // End loop()


// Read "sampleSize" samples and report the average
int ReadAxis(int axisPin)
{
  long reading = 0;
  analogRead(axisPin);
  delay(1);
  for (int i = 0; i < sampleSize; i++)
  {
    reading += analogRead(axisPin);
  }
  return reading/sampleSize;
}
```

```
// Find the extreme raw readings from each axis
void AutoCalibrate(int xRaw, int yRaw, int zRaw)
{
  Serial.println("Calibrate");
  if (xRaw < xRawMin)
  {
    xRawMin = xRaw;
  }
  if (xRaw > xRawMax)
  {
    xRawMax = xRaw;
  }

  if (yRaw < yRawMin)
  {
    yRawMin = yRaw;
  }
  if (yRaw > yRawMax)
  {
    yRawMax = yRaw;
  }

  if (zRaw < zRawMin)
  {
    zRawMin = zRaw;
  }
  if (zRaw > zRawMax)
  {
    zRawMax = zRaw;
  }
}

}
```

APPENDIX J

pwcspZBSensorNode.ino Sensor Module Arduino Sketch

```
/*
 Transmit Tri-Axial Accelerometer Readings to a SD Card

 Used to take tri-axial accelerometer readings, save data to
 SD card. Can also be used to calibrate an Adafruit ADXL335,
 ADXL326, or ADXL377 Analog Accelerometer Breakout Board and
 print out readings to the serial monitor.

 Expected Setup:

 [Adafruit Data Logging Shield]
 | | |
 V V V
 [Arduino Uno]


 Last modified April 16, 2015
 by L'Nard E.T. Tufts II | Student Research Assistant
 */


/*Declare Required Libraries for Sketch*/
#include <SD.h>
#include <Wire.h>
#include "RTClib.h"

/*Declare Variable Names for Pins Used in Sketch*/
const int xInput = A0;//Analog Input Pin 0
const int yInput = A1;//Analog Input Pin 1
const int zInput = A2;//Analog Input Pin 2
const int buttonPin = 2;//Used for calibration

/*****************************************************************/
/****CHANGE BASED ON WHICH SENSOR NODE THIS SKETCH IS UPLOADED TO****/
// Back Plate:   DD = 22
// Chasis Frame:     B1 = 23
// Seat Pan: 9F = 24

int sensortag = 22;
/*****************************************************************/

/* Calibration */
/*
 Raw Ranges:
 If unknown, initialize to mid-range (512) and allow calibration to find
 the minimum and maximum raw values for each axis. Once determined, set
 to calibrated values.
 */
int xRawMin = 406;// 3G: 406  5G: 490
int xRawMax = 608;// 3G: 608  5G: 530

int yRawMin = 400;// 3G: 490  5G: 400
```

```
int yRawMax = 600;// 3G: 530  5G: 600

int zRawMin = 422;// 3G: 422  5G: 495
int zRawMax = 627;// 3G: 627  5G: 535

/* End Calibration */

/* SD Card Logging */
// take multiple samples to reduce noise
const int sampleSize = 1;

// how many milliseconds between entries; grabbing data and logging it
#define LOG_INTERVAL  1// (reduce to take more/faster data)

/*
how many milliseconds before writing the logged data permanently to disk
 set it to the LOG_INTERVAL to write each time (safest)
 set it to 10*LOG_INTERVAL to write all data every 10 datareads
 you could lose up to the last 10 reads if power is lost but it uses less
 power and is much faster! */
#define SYNC_INTERVAL 200// mills between calls to flush() to write data to card
uint32_t syncTime = 0;// time of last sync()

/****SWITCH ON (1) OR OFF (0) BASED ON DESIRED OUTPUTS****/
#define ECHO_TO_SERIAL   0// echo data to serial port
#define WAIT_TO_START    0// wait for serial input in setup()

RTC_DS1307 RTC;// define the Real Time Clock object

// for the data logging shield, we use digital pin 10 for the SD cs line
const int chipSelect = 10;

// the logging file
File logfile;

/*This function returns the date and time for SD card file access and
 modify time. One needs to call in setup() to register this callback
 function: SdFile::dateTimeCallback(file_date_time);*/
void dateTime(uint16_t* date, uint16_t* time)
{
  DateTime now = RTC.now();
  *date=FAT_DATE(now.year(),now.month(),now.day());
  *time=FAT_TIME(now.hour(),now.minute(),now.second());
}
/* End SD Card Logging */

/*This function prints an error message and stops the sketch in the
 event of an  error */
void error(char *str)
{
#if ECHO_TO_SERIAL
  Serial.print("error: ");
```

```
  Serial.println(str);
#endif
  while(1);
}


/*Setup function runs once to initialize pin modes, the serial port, and
 the logging file */
void setup(void){
  analogReference(EXTERNAL); // IMPOTRTANT if AREF is connected to accel
  Serial.begin(115200);
  /*Make sure the Serial.begin(#) baud rate matches the baud rate of all
   XBees, Arduinos, and Processing sketches used in this project*/

#if WAIT_TO_START
  Serial.println("Type any character to start");
  while (!Serial.available());
#endif //WAIT_TO_START

  /*SD Card Setup*/
  // make sure that the default chip select pin is set to output, even
  // if you don't use it:
  pinMode(10, OUTPUT);

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    error("Card failed, or not present");
  }
#if ECHO_TO_SERIAL
  Serial.println("card initialized.");
#endif

  // Attach callback function to provide date time to files
  //SdFile::dateTimeCallback(dateTime);

  // create a new file
  char  filename[]  ="LOGGER00.CSV"; //Optional: change to desired filename
  for (uint8_t i = 0; i < 100; i++) {
    // Change the counter for the file
    filename[6] = i/10 +'0'; // Tens place
    filename[7] = i%10 +'0'; // Ones place
    if  (!SD.exists(filename)) {
      // only open a new file if it doesn't exist
       logfile =SD.open(filename, FILE_WRITE);
      break;  // leave the loop!
    }
  }

  if (! logfile) {
    error("couldnt create file");
  }
```

```
#if ECHO_TO_SERIAL
  Serial.print("Logging to: ");
  Serial.println(filename);
#endif //ECHO_TO_SERIAL

  // connect to the Real Time Clock (RTC)
  Wire.begin();
  if (!RTC.begin()) {
    logfile.println("RTC failed");
#if ECHO_TO_SERIAL
    Serial.println("RTC failed");
#endif  //ECHO_TO_SERIAL
  }

  // print header row to file
  if (sensortag == 22){
    logfile.println("ds_back_t,ds_back_x,ds_back_y,ds_back_z");
  }
  else if (sensortag == 23){
    logfile.println("ds_frame_t,ds_frame_x,ds_frame_y,ds_frame_z");
  }
  else if (sensortag == 24){
    logfile.println("ds_pan_t,ds_pan_x,ds_pan_y,ds_pan_z");
  }
#if ECHO_TO_SERIAL
  Serial.println("millis,a_x,a_y,a_z");
#endif //ECHO_TO_SERIAL
}


/*Loop function repeats consecutively until power is removed or Arduino board is reset.
 Reads, transmits, and logs accelerometer data. Also contains code for calibrating the
 sensor.
 */
void loop (){
  // delay for the amount of time we want between readings
  //delay((LOG_INTERVAL - 1) - (millis() % LOG_INTERVAL));

  /* READ DATA */
  //Call ReadAxis() function to read pins (See Function after loop ())
  int xRaw = ReadAxis(xInput);
  int yRaw = ReadAxis(yInput);
  int zRaw = ReadAxis(zInput);
  uint32_t m =millis(); // note time data was taken (milliseconds since start)
  /* END READ DATA */


  /* CALIBRATION */
  // calibrate during button press using AutoCalibrate function
  if (digitalRead(buttonPin)  ==LOW)
  {
    AutoCalibrate(xRaw, yRaw, zRaw);
```

```
  }
  /* END CALIBRATION */

  /* RAW CONVERSION */
  // Convert raw values to 'milli-Gs"
  long xScaled  =map(xRaw, xRawMin, xRawMax, -1000, 1000);
  long yScaled  =map(yRaw, yRawMin, yRawMax, -1000, 1000);
  long zScaled  =map(zRaw, zRawMin, zRawMax, -1000, 1000);

  // re-scale to m/s^2
  float xAccel = (xScaled / 1000.0)*(-9.81);
  float yAccel = (yScaled / 1000.0)*(-9.81);
  float zAccel = (zScaled / 1000.0)*(-9.81);
  /* END RAW CONVERSION */

  /* DATA LOGGING */
  /* writes values to SD logging file as comma separated values */

  // Read Time

  logfile.print(m);
  logfile.print(",");

  // Acceleration Values

  logfile.print(xAccel);
  logfile.print(", ");
  logfile.print(yAccel);
  logfile.print(", ");
  logfile.println(zAccel); // End line


  // Now we write data to disk! Don't sync too often - requires 2048 bytes of I/O to SD ca
  // which uses a bunch of power and takes time
  if ((millis() - syncTime) < SYNC_INTERVAL)return;
   syncTime  =millis();

  logfile.flush();

  /* END DATA LOGGING */
} // End loop()


// Read "sampleSize" samples and report the average
int ReadAxis(int axisPin)
{
  long reading = 0;
  analogRead(axisPin);
  delay(1);
  for (int i = 0; i < sampleSize; i++)
  {
     reading +=analogRead(axisPin);
```

```
  }
  return reading/sampleSize;
}


// Find the extreme raw readings from each axis
void AutoCalibrate(int  xRaw,int  yRaw,int zRaw)
{
  Serial.println("Calibrate");
  if (xRaw < xRawMin)
  {
    xRawMin = xRaw;
  }
  if (xRaw > xRawMax)
  {
    xRawMax = xRaw;
  }

  if (yRaw < yRawMin)
  {
    yRawMin = yRaw;
  }
  if (yRaw > yRawMax)
  {
    yRawMax = yRaw;
  }

  if (zRaw < zRawMin)
  {
    zRawMin = zRaw;
  }
  if (zRaw > zRawMax)
  {
    zRawMax = zRaw;
  }
}
```

APPENDIX K

pwcspZBBaseStation.ino Arduino Sketch

```
/*
 Receive Tri-Axial Accelerometer Frame Data from Multiple Router XBees

 Decodes XBee API frame data consisting of Tri-Axial Accelerometer readings.

 Used with XBee radio configured as follows:
 ZigBee Coordinator API
 BD - 115200 [7]
 NB - N [0]
 SB - 2 [1]
 AP - 2

 Full profile can be viewed at:
 smb://shrs-p01/users/pwcsp/Development/Configuration/profile_xb_coordinator.xml

 Last modified January 7, 2015
 by L'Nard E.T. Tufts II | Student Research Assistant
 */


/*Declare Variables and Pins Used in Sketch*/
int debugLED = 13; //Embedded LED attached to digital pin 13
int routerXBTag;
int analogValuex = 0;
int analogValuey = 0;
int analogValuez = 0;
byte source = 0x00;

/*Setup function runs once to initialize pin modes and the serial port*/
void setup() {
  pinMode(debugLED,OUTPUT);
  Serial.begin(115200);
  /*Make sure the Serial.begin(#) baud rate matches the baud rate of all XBees,
   Arduinos, and Processing sketches used in this project*/
}


/*
Loop function repeats consecutively until power is removed or Arduino board is reset.
The loop will read through a received API frame, note which router station it was sent
from, and pull out the analog accelerometer data. It then sends the relevant data to
the computer's serial port to be analyzed or visulaized using another program i.e. Pro
cessing
 */
void loop() {

  // make sure everything we need is in the serial buffer. API fram should be 22 bytes
long
  if (Serial.available() >= 21) {

    // look for the start byte to ensure we're not in the midle of a frame
    // the start of each frame begins with the byte: 0x7E
    if (Serial.read() == 0x7E) {

      //blink debug LED to indicate when data is received
      digitalWrite(debugLED, HIGH);
      delay(10);
      digitalWrite(debugLED, LOW);

      // read the variables that we're not using out of the buffer
```

```
      for (int i = 0; i<11; i++) {
        byte discard = Serial.read();
      }
      //check the last byte of the source address in order to identify which router
XBee sent the data
      source = Serial.read();

      if (source == 0xDD) { // used for arm rest readings
        routerXBTag = 2000;
      }
      else if (source == 0xB1) { // used for wheelchair frame readings
        routerXBTag = 2001;
      }
      else if (source == 0x9F) { // used for foot rest readings
        routerXBTag = 2002;
      }

      // read the variables that we're not using out of the buffer
      for (int j = 0; j<7; j++) {
        byte discard = Serial.read();
      }

      // the next 6 bytes are the X, Y, and Z accelerometer readings broken into a
high and low byte
      int analogHighx = Serial.read();
      int analogLowx = Serial.read();
      int analogHighy = Serial.read();
      int analogLowy = Serial.read();
      int analogHighz = Serial.read();
      int analogLowz = Serial.read();

      // reassemble high and low byte through arithmetic conversion to its 10-bit
decimal value
      analogValuex = analogLowx + (analogHighx * 256);
      analogValuey = analogLowy + (analogHighy * 256);
      analogValuez = analogLowz + (analogHighz * 256);

      // send X, Y, and Z readings along with a tag of the sender XBee to serial port
as comma separated
      // values to be read by external program (Processing, MATLab, LabVIEW, etc.)
      Serial.print (analogValuex);
      Serial.print (", ");
      Serial.print (analogValuey);
      Serial.print (", ");
      Serial.print (analogValuez);
      Serial.print (", ");
      Serial.println (routerXBTag);
    }
  }
  Serial.flush(); // waits for the transmission of outgoing serial data to complete
}

}
```

APPENDIX L

pwcspMultiSensorPlot.pde Processsing Sketch

```
/*****************************************************************************/
/*            Graph Data from Multiple Multi-Axis Sensors in Processing       */
/*****************************************************************************/
/*    Takes ASCII-encoded strings from serial port and graphs them. Expects COMMA or   */
/*   TAB SEPARATED values, followed by a newline, or newline and carriage return. Can  */
/*         read 10-bit values from Arduino, 0-1023 (or even higher if you wish)       */
/*****************************************************************************/
/* Last modified January 7, 2015                                             */
/* by L'Nard Tufts | Student Research Assistant                              */
/* adapted from Eric Forman | www.ericforman.com | teaching.ericforman.com   */
/*****************************************************************************/


/* Declare and Initialize Variables and Ports*/

import processing.serial.*;
Serial myPort;

/* change this to match how many values your Arduino is sending */
int numValues = 3; // number of inputs per sensor i.e. 3 for a tri-axial accelerometer
int numSensors = 3; // number of sensor nodes (in this case 3 named b, d, and f)

/* initialize arrays (add or reduce copies based on numSensors) */
float[] values_b   = new float[numValues];
int[]   min_b      = new int[numValues];
int[]   max_b      = new int[numValues];
color[] valColor_b = new color[numValues];

float[] values_d   = new float[numValues];
int[]   min_d      = new int[numValues];
int[]   max_d      = new int[numValues];
color[] valColor_d = new color[numValues];

float[] values_f   = new float[numValues];
int[]   min_f      = new int[numValues];
int[]   max_f      = new int[numValues];
color[] valColor_f = new color[numValues];

float partH;          // partial screen height

float xPos_b = 0; // horizontal position of the b graph
float xPos_d = 0; // horizontal position of the d graph
float xPos_f = 0; // horizontal position of the f graph

String token; // sensor token to identify source node

char axis; // acceleration axis i.e. x, y, or z

float k = 0; // counter
int a = 0; // counter

/* Setup funtion runs once to initialize key values and window settings */
void setup() {
  size(1680, 970); // set window size in pixels
  partH = height / (numValues * numSensors); // divide screen by number of data streams
```

```
/*****************************************************************************/
/* CHECK THE NAME OF THE SERIAL PORT YOUR BASE STATION ARDUINO IS CONNECTED TO AND */
/* MAKE SURE BAUD_RATE MATCHES THE BAUD RATE OF ALL XBEES, ARDUINOS, AND PROCESSING*/
/*     SKETCHS USED IN THIS PROJECT. ENTER PORT NAME and BAUD RATE AS FOLLOWS:     */
/*            myPort = new Serial(this, "SERIAL_PORT_NAME", BAUD_RATE)             */
/*****************************************************************************/
myPort = new Serial(this, "/dev/tty.usbmodem1411", 115200);

// don't generate a serialEvent() until you get a newline character:
myPort.bufferUntil('\n');

textSize(10);

background(0); // set background to black
noStroke();
noLoop();

/*****************************************************************************/
/*                                  INITIALIZE                                 */
/*  EDIT THESE TO MATCH HOW MANY VALUES YOU ARE READING AND WHAT COLORS YOU LIKE   */
/*  min and max: 8-bit (0-255)                                                 */
/*               10-bit (0-1023)                                               */
/*               1-bit (0-1) e.g. a digital switch                             */
/*               custom range                                                  */
/*  color mode is RGB by default                                               */
/*****************************************************************************/
// node d : x-axis
values_d[0] = 0;
min_d[0] = 0;
max_d[0] = 1023;
valColor_d[0] = color(255, 0, 0); // red

// node d : y-axis
values_d[1] = 0;
min_d[1] = 0;
max_d[1] = 1023;
valColor_d[1] = color(0, 255, 0); // green

// node d : z-axis
values_d[2] = 0;
min_d[2] = 0;
max_d[2] = 1023;
valColor_d[2] = color(0, 0, 255); // blue

// node b : x-axis
values_b[0] = 0;
min_b[0] = 0;
max_b[0] = 1023;
valColor_b[0] = color(0, 255, 255); // cyan

// node b : y-axis
values_b[1] = 0;
min_b[1] = 0;
max_b[1] = 1023;
valColor_b[1] = color(255, 0, 255); // magenta

// node b : z-axis
```

```
  values_b[2] = 0;
  min_b[2] = 0;
  max_b[2] = 1023;
  valColor_b[2] = color(255, 255, 0); // yellow

  // node f : x-axis
  values_f[0] = 0;
  min_f[0] = 0;
  max_f[0] = 1023;
  valColor_f[0] = color(0, 127, 255); // teal

  // node f : y-axis
  values_f[1] = 0;
  min_f[1] = 0;
  max_f[1] = 1023;
  valColor_f[1] = color(255, 0, 127); // purple

  // node f : z-axis
  values_f[2] = 0;
  min_f[2] = 0;
  max_f[2] = 1023;
  valColor_f[2] = color(127, 255, 0); // puke
}


void draw() {
  // in this sketch, everything happens inside serialEvent()
  // but you can also do stuff in every frame if you wish
}


void serialEvent(Serial myPort) { // runs each time a data string is received
  float t = 0; // counter

  String inString = myPort.readStringUntil('\n'); // get the ASCII string

  // associate the horizontal pixel position of the data with time i.e. plot in real time
  t = (millis()%(width*100));
  k = map(t, 0, (width*100), 0, width);

  xPos_d = k-a;
  xPos_b = k-a;
  xPos_f = k-a;

  /* draw horizontal gridlines for context */
  float section = 4;
  for (float m = 1; m < (numValues*numSensors)*section+1; m++) {
    stroke(30); // color of quarter grid lines, dark grey
    if (m%section==0) { // color of axis division lines, light grey
      stroke(125);
    }
    if (m%(section*numSensors)==0) { // color of node division lines, orangish
      stroke(125,20,0);
    }
    line(0, m*(height/((numValues*numSensors)*section)), width, m*(height/
((numValues*numSensors)*section)));
  }
```

```
//print("raw: \t" + inString);  // < - uncomment this to debug serial input from Arduino

if (inString != null) { // expects: analogValuex, analogValuey, analogValuez, routerXBTag
  // trim off any whitespace: analogValuex,analogValuey,analogValuez,routerXBTag
  inString = trim(inString);
  // split the string on the delimiters and convert the resulting substrings into an array:
  int[] inArray = int(splitTokens(inString, ", \t")); // delimiter can be comma space or tab

  if (inArray.length == 1+numValues) { // make sure we have all the expected values
    /* Carry out commands based on which node sent the data. Each case is identical */
    /* except acts on variables corresponding to the current transmitting node      */
    switch(inArray[3]) {
    case 2000:
      values_d = float(shorten(inArray)); // trim routerXBTag off the array
      token = "40B554DD"; // set the sensor node token

      // if the array has at least the # of elements as your # of sensors, you know
      // you got the whole data packet.  Map the numbers and put into the variables:
      if (values_d.length >= numValues) {
        // iterate over each cell of the values_d array
        for (int i=0; i<numValues; i++) {
          if (i == 0) {
            axis = 'X';
          } else if (i == 1) {
            axis = 'Y';
          } else if (i == 2) {
            axis = 'Z';
          }

          // print values in key box:
          fill(50);
          noStroke();
          rect(0, partH*i+1, 120, 12);
          fill(255);
          text(int(values_d[i]), 2, partH*i+10);
          fill(125);
          text(token, 40, partH*i+10);
          text(axis, 105, partH*i+10);

          // map to the range of partial screen height:
          float scaled_d = map(values_d[i], 200, 823, 0, partH);

          if ((values_d[i] > 0) && (values_d[i] < 1023)) { // ignore extremes/out of bounds
            //draw data:
            /*
            // plot histogram lines
            stroke(valColor_d[i]);
            line(xPos_d, partH*(i+1), xPos_d, partH*(i+1) - scaled_d);
            */

            // scatter plot
            stroke(valColor_d[i]);
            fill(valColor_d[i]);
            ellipse(xPos_d, partH*(i+1) - scaled_d, 1, 1); // draw circle for point
          }

          //println("\t"+values_d[i]);   // <- uncomment this to debug values
        }
```

```
    //println();                  // <- uncomment this to debug values

    // if at the edge of the screen, go back to the beginning:
    if (xPos_d >= .97*width) {
      background(0);  // erase screen with black
      a++; // increase counter for timing/x-position calculation
    }
  }
  break;

case 2001:
  values_b = float(shorten(inArray));
  token = "40B554B1";
  if (values_b.length >= numValues) {
    for (int i=0; i<numValues; i++) {
      if (i == 0) {
        axis = 'X';
      } else if (i == 1) {
        axis = 'Y';
      } else if (i == 2) {
        axis = 'Z';
      }

      fill(50);
      noStroke();
      rect(0, partH*(i+numValues)+1, 120, 12);
      fill(255);
      text(int(values_b[i]), 2, partH*(i+numValues)+10);
      fill(125);
      text(token, 40, partH*(i+numValues)+10);
      text(axis, 105, partH*(i+numValues)+10);

      // map to the range of partial screen height:
      float scaled_b = map(values_b[i], 200, 823, 0, partH);

      if ((values_b[i] > 0) && (values_b[i] < 1023)) {
        /*
        stroke(valColor_b[i]);
        line(xPos_b, partH*((i+numValues)+1), xPos_b, partH*((i+numValues)+1) - scaled_b);
        */
        stroke(valColor_b[i]);
        fill(valColor_b[i]);
        ellipse(xPos_b, partH*((i+numValues)+1) - scaled_b, 1, 1);
      }
      //println("\t"+values_b[i]);   // <- uncomment this to debug values
      //println();                   // <- uncomment this to debug values

      // if at the edge of the screen, go back to the beginning:
      if (xPos_b >= .97*width) {
        background(0);           // erase screen with black
        a++;
      }
    }
  }
  break;

case 2002:
  values_f = float(shorten(inArray));
```

```
        token = "40B5B89F";
        if (values_f.length >= numValues) {
          for (int i=0; i<numValues; i++) {
            if (i == 0) {
              axis = 'X';
            } else if (i == 1) {
              axis = 'Y';
            } else if (i == 2) {
              axis = 'Z';
            }

            fill(50);
            noStroke();
            rect(0, partH*(i+(2*numValues))+1, 120, 12);
            fill(255);
            text(int(values_f[i]), 2, partH*(i+(2*numValues))+10);
            fill(125);
            text(token, 40, partH*(i+(2*numValues))+10);
            text(axis, 105, partH*(i+(2*numValues))+10);

            // map to the range of partial screen height:
            float scaled_f = map(values_f[i], 200, 823, 0, partH);

            if ((values_f[i] > 0) && (values_f[i] < 1023)) {
              /*
              stroke(valColor_f[i]);
              line(xPos_f, partH*((i+(2*numValues))+1), xPos_f, partH*((i+(2*numValues))+1) -
scaled_f);
              */
              stroke(valColor_f[i]);
              fill(valColor_f[i]);
              ellipse(xPos_f, partH*((i+(2*numValues))+1) - scaled_f, 1, 1);
            }
            //println("\t"+values_f[i]);   // <- uncomment this to debug values
            //println();                   // <- uncomment this to debug values

            // if at the edge of the screen, go back to the beginning:
            if (xPos_f >= .97*width) {
              background(0);           // erase screen with black
              a++;
            }
          }
        }
        break;
      } // end switch case
    } // end length if
  } // end !=null if
  redraw();
} // end serial event
```

APPENDIX M

Wheelchair Skills Test (WST) Version 4.2 Form

Wheelchair Skills Test (WST) Version 4.2 Form
Powered Wheelchairs Operated by Their Users
Name of wheelchair user: _____
Tester: _____ Date: _____

#	Individual Skill	Capacity Score* (0-2)	Training Goal? (Y/N)	Comments
1	Moves controller/tiller away and back			
2	Turns controller on and off			
3	Selects drive modes and speeds			
4	Operates body positioning options			
5	Disengages and engages motors			
6	Operates battery charger			
7	Rolls forwards (10 m)			
8	Rolls backwards (2 m)			
9	Turns while moving forwards (90°)			
10	Turns while moving backwards (90°)			
11	Turns in place (180°)			
12	Maneuvers sideways (0.5 m)			
13	Gets through hinged door			
14	Reaches high object (1.5 m)			
15	Picks object up from floor			
16	Relieves weight from buttocks (3 sec)			
17	Transfers to and from bench			
18	Rolls 100 m			
19	Avoids moving obstacles			
20	Ascends 5° incline			
21	Descends 5° incline			
22	Ascends 10° incline			
23	Descends 10° incline			
24	Rolls across side-slope (5°)			
25	Rolls on soft surface (2 m)			
26	Gets over gap (15 cm)			
27	Gets over threshold (2 cm)			
28	Ascends low curb (5 cm)			
29	Descends low curb (5 cm)			
30	Gets from ground into wheelchair			
	Total score:*	%		

* See score options and formula for calculating total score on page 2

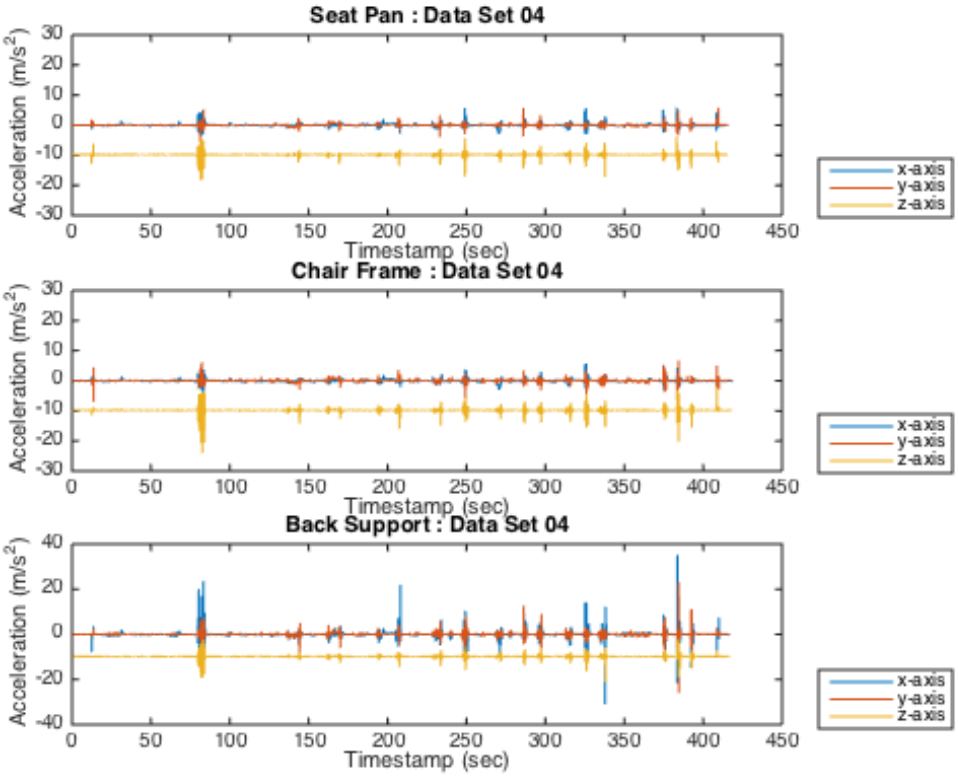
APPENDIX N

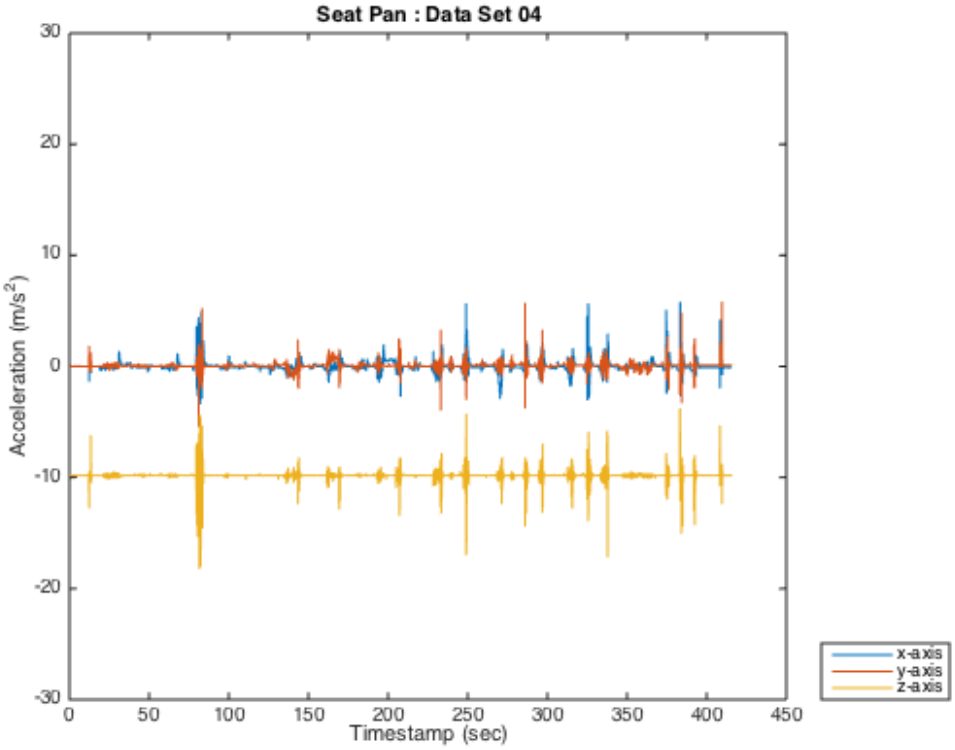
MATLab Data Processing and Analysis Script Data Set 04

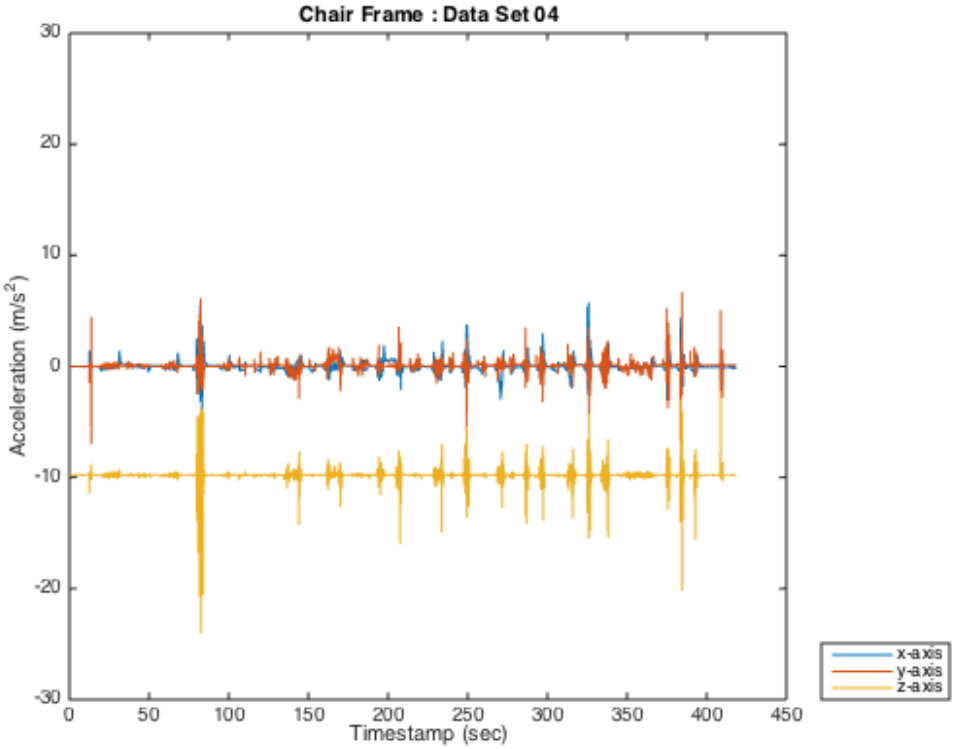
```
close all;
load('DataSet04_v2.mat')

x_b_off = -.39;
y_b_off = +.10;
z_b_off = -.77;

x_p_off = +.19;
y_p_off = +.69;
z_p_off = +.77;



x_f_off = +.28;
y_f_off = +.54;
z_f_off = +.14;



x_b = -((ds4_back_z) + (z_b_off));
y_b = ds4_back_x + x_b_off;
z_b = -((ds4_back_y) + (y_b_off));
t_b = ds4_back_t/1000;

x_p = ds4_pan_y + y_p_off;
y_p = ds4_pan_x + (x_p_off);
z_p = -((ds4_pan_z)+ (z_p_off));
t_p = ds4_pan_t/1000;

x_f = -((ds4_frame_y) + (y_f_off));
y_f = ds4_frame_x + x_f_off;
z_f = ds4_frame_z + z_f_off;
t_f = ds4_frame_t/1000;
figure(1)
subplot (3,1,1)
plot(t_p,x_p,t_p,y_p,t_p,z_p)
ylim([-30 30])
```

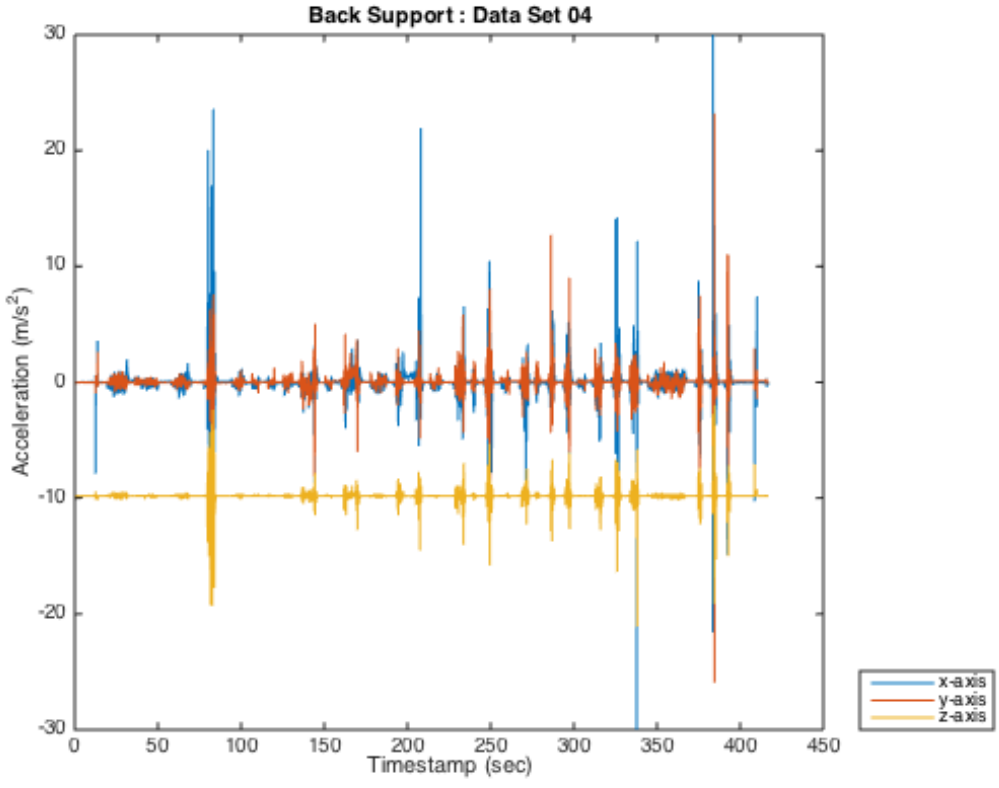
```
title('Seat Pan : Data Set 04')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')

subplot (3,1,2)
plot(t_f,x_f,t_f,y_f,t_f,z_f)
ylim([-30 30])
title('Chair Frame : Data Set 04')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')

subplot (3,1,3)
ylim([-30 30])
plot(t_b,x_b,t_b,y_b,t_b,z_b)
title('Back Support : Data Set 04')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')

figure(2)
plot(t_p,x_p,t_p,y_p,t_p,z_p)
ylim([-30 30])
title('Seat Pan : Data Set 04')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')

figure(3)
plot(t_f,x_f,t_f,y_f,t_f,z_f)
ylim([-30 30])
title('Chair Frame : Data Set 04')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')

figure(4)
```

```
plot(t_b,x_b,t_b,y_b,t_b,z_b)
ylim([-30 30])
title('Back Support : Data Set 04')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')
hold off
```


Seat Pan : Data Set 04
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis
Chair Frame : Data Set 04
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis
Back Support : Data Set 04
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis


Seat Pan : Data Set 04
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis


Chair Frame : Data Set 04
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis


Back Support : Data Set 04
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis

```
%m = 1
```

```
back_x_mat = zeros(length(t_b),2);
back_x_mat(:,1) = t_b';
back_x_mat(:,2) = x_b';

%m = 2
back_y_mat = zeros(length(t_b),2);
back_y_mat(:,1) = t_b';
back_y_mat(:,2) = y_b';

%m = 3
back_z_mat = zeros(length(t_b),2);
back_z_mat(:,1) = t_b';
back_z_mat(:,2) = z_b';

%m = 4
frame_x_mat = zeros(length(t_f),2);
frame_x_mat(:,1) = t_f';
frame_x_mat(:,2) = x_f';

%m = 5
frame_y_mat = zeros(length(t_f),2);
frame_y_mat(:,1) = t_f';
frame_y_mat(:,2) = y_f';

%m = 6
frame_z_mat = zeros(length(t_f),2);
frame_z_mat(:,1) = t_f';
frame_z_mat(:,2) = z_f';

%m = 7
pan_x_mat = zeros(length(t_p),2);
pan_x_mat(:,1) = t_p';
pan_x_mat(:,2) = x_p';

%m = 8
pan_y_mat = zeros(length(t_p),2);
pan_y_mat(:,1) = t_p';
pan_y_mat(:,2) = y_p';
```

```
%m = 9
pan_z_mat = zeros(length(t_p),2);
pan_z_mat(:,1) = t_p';
pan_z_mat(:,2) = z_p';
disp(' ');
disp('   ');
disp(' ');
disp('  Based on th_weighted.m  ver 1.2  May 31, 2013 by Tom Irvine  Email:
tom@vibrationdata.com ');
disp('  Adapthed by LNard Tufts  March 24, 2015 ');
disp('  ');
disp('  ');
disp('  This program converts an acceleration time history to a ');
disp('  weighted format per ISO 2631. ');
disp('  ');
disp(' ');
%
for a=1:3
    for m=1:3
        clear amp;
        clear f;
        clear length;
        clear THM;
        clear t;
        clear f;
        clear y;
        clear ww;
        clear matrix_name;

        fig_num=5;
        %[t,f,dt,sr,tmx,tmi,~,ncontinue]=enter_time_history();
        %
        %disp(' Select file input method ');
        %disp('   1=external ASCII file ');
        %disp('   2=file preloaded into Matlab ');
        %disp('   3=Excel file ');
        %file_choice = input('');
```

```
        file_choice=2;
        %
        if(file_choice==1)
            [filename, pathname] = uigetfile('*.*');
            filename = fullfile(pathname, filename);
            fid = fopen(filename,'r');
            THM = fscanf(fid,'%g %g',[2 inf]);
            THM=THM';
        end
        if(file_choice==2)
            %FS = input(' Enter the matrix name:  ','s');
            if a==1;
                clear THM;
                name={'Back Plate: X-Axis';'Back Plate: Y-Axis';'Back Plate: Z-Axis'};
                kxh=0.8; kyh=0.5; kzh=0.4;
                kxc=0.8; kyc=0.5; kzc=0.4;

                matrix_b(:,:,1) = back_x_mat;
                start_time(1) = 13.64;
                end_time(1) = 408.7;
                weighting(1) = 4;

                matrix_b(:,:,2) = back_y_mat;
                start_time(2) = 13.64;
                end_time(2) = 408.7;
                weighting(2) = 2;

                matrix_b(:,:,3) = back_z_mat;
                start_time(3) = 13.64;
                end_time(3) = 408.7;
                weighting(3) = 2;
                THM = matrix_b(:,:,m);
            elseif a==2;
                clear THM;
                name={'Chassis Frame: X-Axis';'Chassis Frame: Y-Axis';'Chassis Frame:
Z-Axis'};
                kxh=1.4; kyh=1.4; kzh=1;
                kxc=1; kyc=1; kzc=1;
```

```
        matrix_f(:,:,1) = frame_x_mat;
        start_time(1) = 13.66;
        end_time(1) = 408.6;
        weighting(1) = 2;

        matrix_f(:,:,2) = frame_y_mat;
        start_time(2) = 13.7;
        end_time(2) = 408.6;
        weighting(2) = 2;

        matrix_f(:,:,3) = frame_z_mat;
        start_time(3) = 13.66;
        end_time(3) = 408.6;
        weighting(3) = 1;
        THM = matrix_f(:,:,m);
    elseif a==3;
        clear THM;
        name={'Seat Pan: X-Axis';'Seat Pan: Y-Axis';'Seat Pan: Z-Axis'};
        kxh=1.4; kyh=1.4; kzh=1;
        kxc=1; kyc=1; kzc=1;

        matrix_p(:,:,1) = pan_x_mat;
        start_time(1) = 13.5;
        end_time(1) = 408.3;
        weighting(1) = 2;

        matrix_p(:,:,2) = pan_y_mat;
        start_time(2) = 13.46;
        end_time(2) = 408.3;
        weighting(2) = 2;

        matrix_p(:,:,3) = pan_z_mat;
        start_time(3) = 13.7;
        end_time(3) = 408.3;
        weighting(3) = 1;
        THM = matrix_p(:,:,m);
    end
```

```
    %HM=evalin('caller',FS);
    %THM = FS;
end
if(file_choice==3)
    [filename, pathname] = uigetfile('*.*');
    xfile = fullfile(pathname, filename);
    %
    THM = xlsread(xfile);
    %
end
t=THM(:,1);
f=THM(:,2);


%
tmx=max(t);
tmi=min(t);
n = length(f);
%dt=(tmx-tmi)/(n-1);
for i=1:(n-1)
    dt_run(i,1)=t(i+1)-t(i);
end
dt=mean(dt_run);
sr=1./dt;
%


disp('****************************************************************')
disp(' ')


disp(name{m})


disp(' ')
disp(' Time Step ');
dtmin=min(diff(t));
dtmax=max(diff(t));
%
out4 = sprintf(' dtmin  = %8.4g sec  ',dtmin);
out5 = sprintf(' dt     = %8.4g sec  ',dt);
out6 = sprintf(' dtmax  = %8.4g sec  ',dtmax);
disp(out4)
disp(out5)
disp(out6)
```

```
        %
        disp(' ')
        disp(' Sample Rate ');
        out4 = sprintf(' srmin  = %8.4g samples/sec  ',1/dtmax);
        out5 = sprintf(' sr     = %8.4g samples/sec  ',1/dt);
        out6 = sprintf(' srmax  = %8.4g samples/sec  \n',1/dtmin);
        disp(out4)
        disp(out5)
        disp(out6)
        %
        ncontinue=1;
        if(((dtmax-dtmin)/dt)>0.01)
            %disp(' ')
            %disp(' Warning:  time step is not constant.  Continue calculation? 1=yes
2=no ')
            %ncontinue=input(' ');
            ncontinue=1;
        end
```

```
************************************************************

Back Plate: X-Axis

 Time Step
 dtmin  =    0.038 sec
 dt     =  0.04234 sec
 dtmax  =     0.27 sec

 Sample Rate
 srmin  =    3.704 samples/sec
 sr     =    23.62 samples/sec
 srmax  =    26.32 samples/sec

************************************************************

Back Plate: Y-Axis

 Time Step
 dtmin  =    0.038 sec
```

```
 dt     =  0.04234 sec
 dtmax  =     0.27 sec

 Sample Rate
 srmin  =    3.704 samples/sec
 sr     =    23.62 samples/sec
 srmax  =    26.32 samples/sec

************************************************************

Back Plate: Z-Axis

 Time Step
 dtmin  =    0.038 sec
 dt     =  0.04234 sec
 dtmax  =     0.27 sec

 Sample Rate
 srmin  =    3.704 samples/sec
 sr     =    23.62 samples/sec
 srmax  =    26.32 samples/sec

************************************************************

Chassis Frame: X-Axis

 Time Step
 dtmin  =    0.029 sec
 dt     =   0.0383 sec
 dtmax  =     0.25 sec

 Sample Rate
 srmin  =        4 samples/sec
 sr     =    26.11 samples/sec
 srmax  =    34.48 samples/sec

************************************************************
```

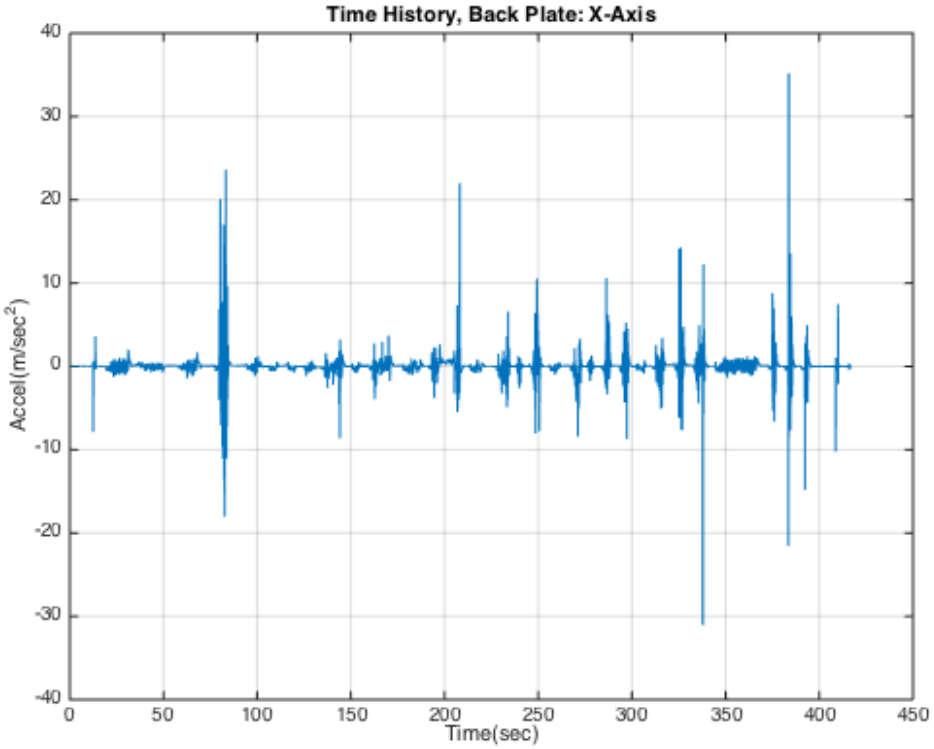
```
Chassis Frame: Y-Axis

 Time Step
 dtmin  =    0.029 sec
 dt     =   0.0383 sec
 dtmax  =     0.25 sec

 Sample Rate
 srmin  =        4 samples/sec
 sr     =    26.11 samples/sec
 srmax  =    34.48 samples/sec

*************************************************************

Chassis Frame: Z-Axis

 Time Step
 dtmin  =    0.029 sec
 dt     =   0.0383 sec
 dtmax  =     0.25 sec

 Sample Rate
 srmin  =        4 samples/sec
 sr     =    26.11 samples/sec
 srmax  =    34.48 samples/sec

*************************************************************

Seat Pan: X-Axis

 Time Step
 dtmin  =    0.028 sec
 dt     =  0.03365 sec
 dtmax  =    0.191 sec

 Sample Rate
 srmin  =    5.236 samples/sec
 sr     =    29.72 samples/sec
```

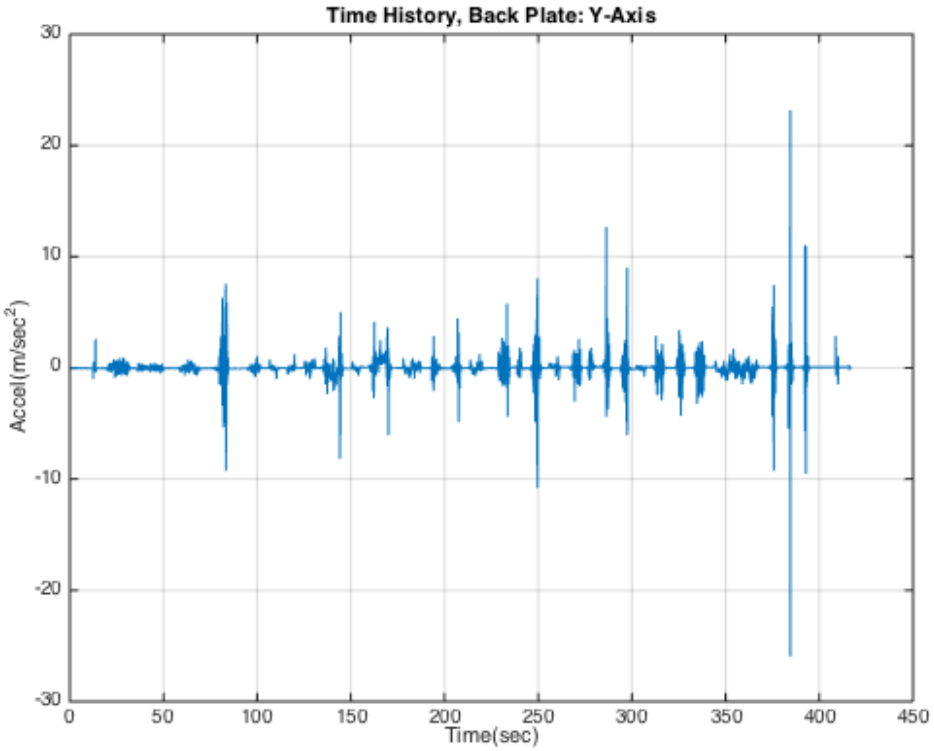
```
 srmax  =    35.71 samples/sec

*****************************************************************

Seat Pan: Y-Axis

 Time Step
 dtmin  =    0.028 sec
 dt     =  0.03365 sec
 dtmax  =    0.191 sec

 Sample Rate
 srmin  =    5.236 samples/sec
 sr     =    29.72 samples/sec
 srmax  =    35.71 samples/sec

*****************************************************************

Seat Pan: Z-Axis

 Time Step
 dtmin  =    0.028 sec
 dt     =  0.03365 sec
 dtmax  =    0.191 sec

 Sample Rate
 srmin  =    5.236 samples/sec
 sr     =    29.72 samples/sec
 srmax  =    35.71 samples/sec
```

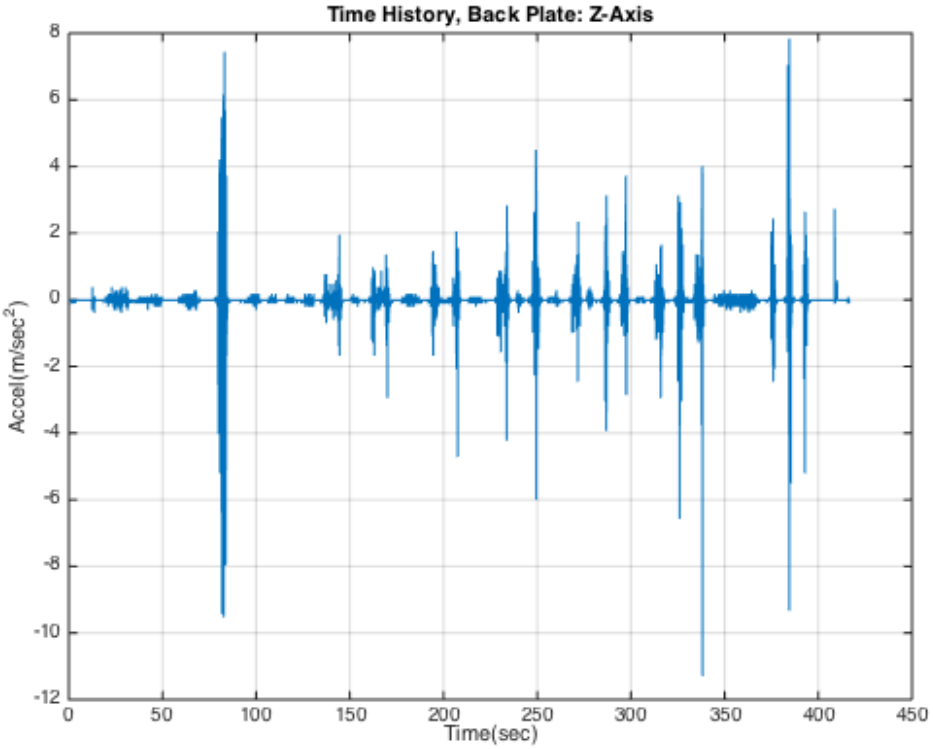
```
        %disp(' ');
        %disp(' Remove mean?  1=yes 2=no');
        %
        %imr=input(' ');
        imr=1;
        if(imr==1)
            f=f-mean(f);
        end
```

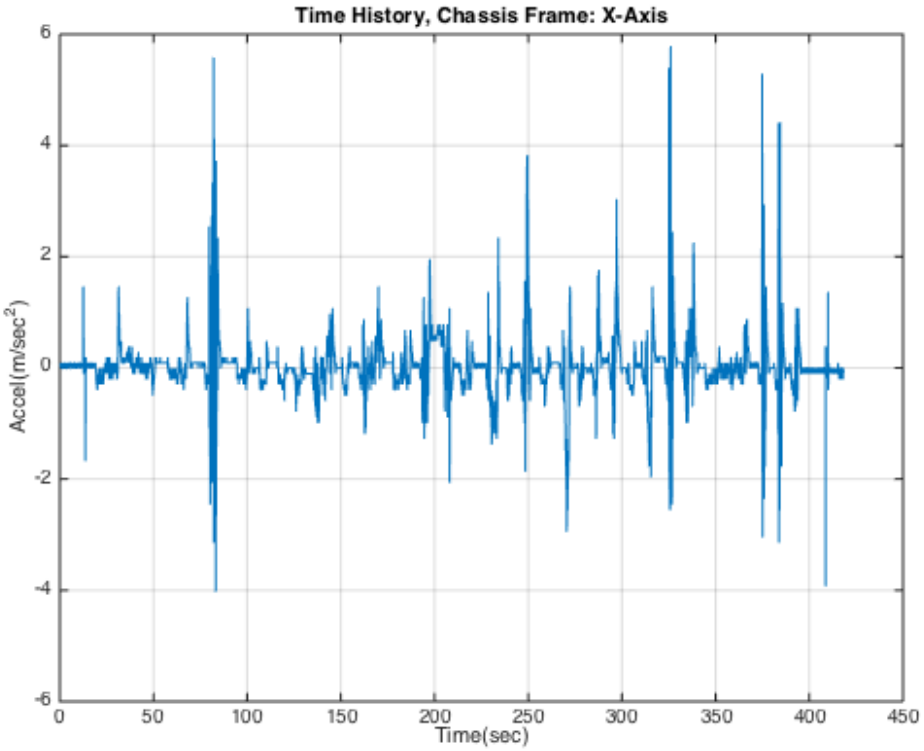
```
%
THM=[t f];
%
iunits=2;
%
while( iunits ~= 1 && iunits ~=2 )
    %
    out1=sprintf('\n Enter input unit:\n  1=G  2= m/sec^2  ');
    disp(out1);
    iunits=input(' ');
    %
end
%
if(iunits==1)
    p_unit=sprintf('G');
else
    p_unit=sprintf('m/sec^2');
end
%
x_label=sprintf('Time(sec)');
y_label=sprintf('Accel(%s)',p_unit);
t_string=sprintf(['Time History, ',name{m}]);
[fig_num]=plot_TH(fig_num,x_label,y_label,t_string,THM);
%
%disp(' ');
%st=input(' Enter starting time (sec) ');
st = start_time(m);
%
%disp(' ');
%te=input(' Enter ending time (sec) ');
te = end_time(m);
%
j=1;
jfirst=1;
jlast=max(size(THM));
for i=1:max(size(THM))
    if(THM(i,1)<st)
        jfirst=i;
```

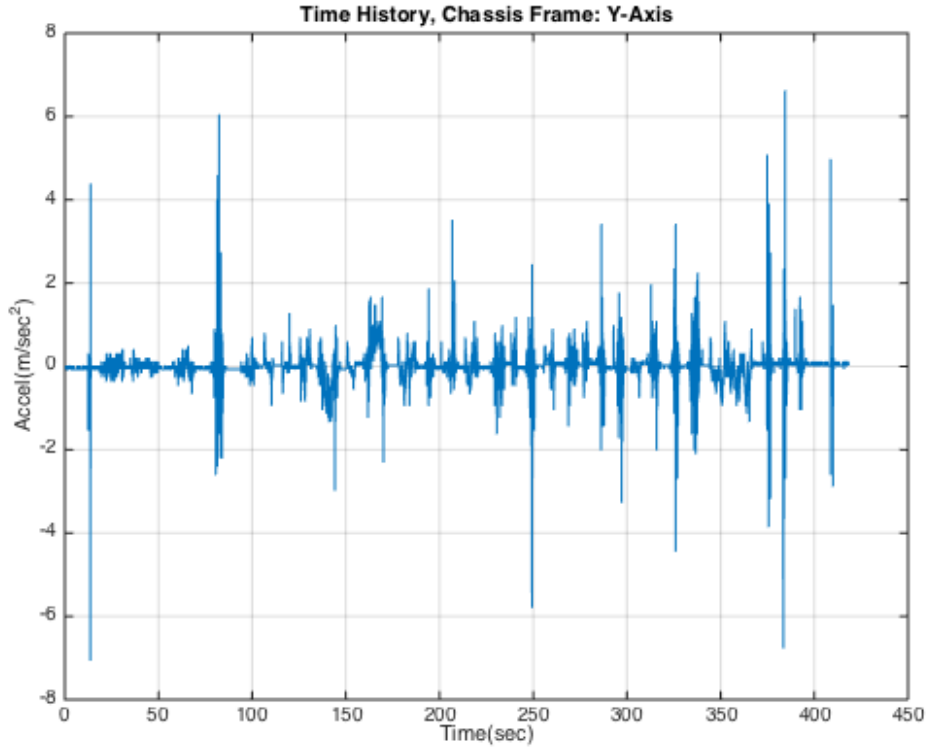
```
        end
        if(THM(i,1)>te)
            jlast=i;
            break;
        end
    end
    %
    tim=double(THM(jfirst:jlast,1));
    amp=double(THM(jfirst:jlast,2));
    %
    if(iunits==1)
        amp=amp*9.81;
    end
    %
    if(imr==1)
        amp=amp-mean(amp);
    end
```

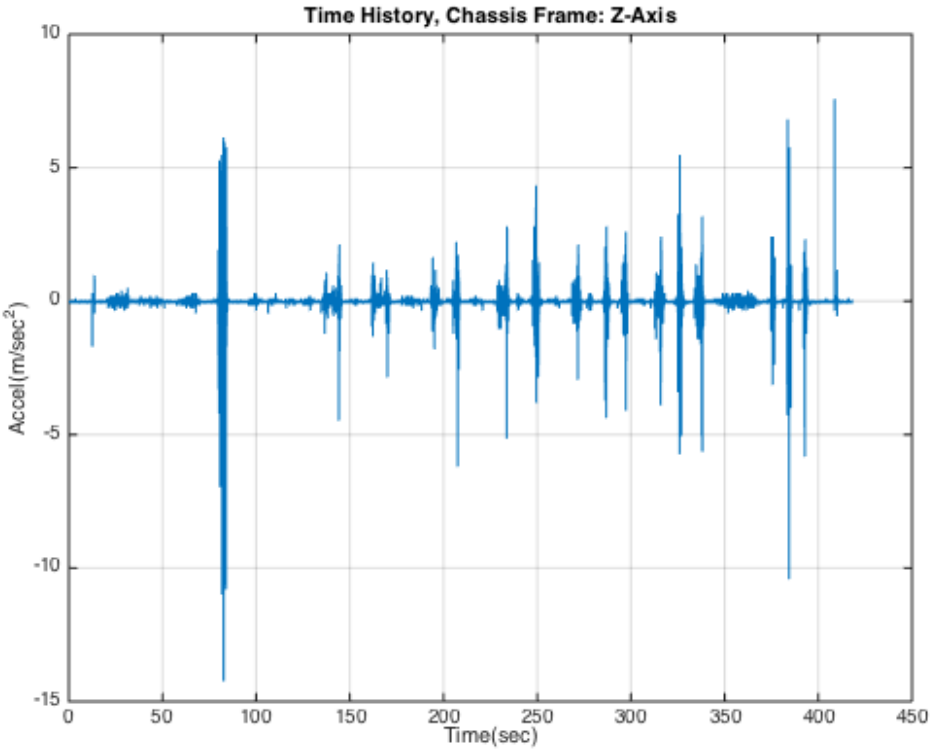

Time History, Back Plate: X-Axis
Accel(m/sec^2)
Time(sec)
40
30
20
10
0
-10
-20
-30
-40
0
50
100
150
200
250
300
350
400
450

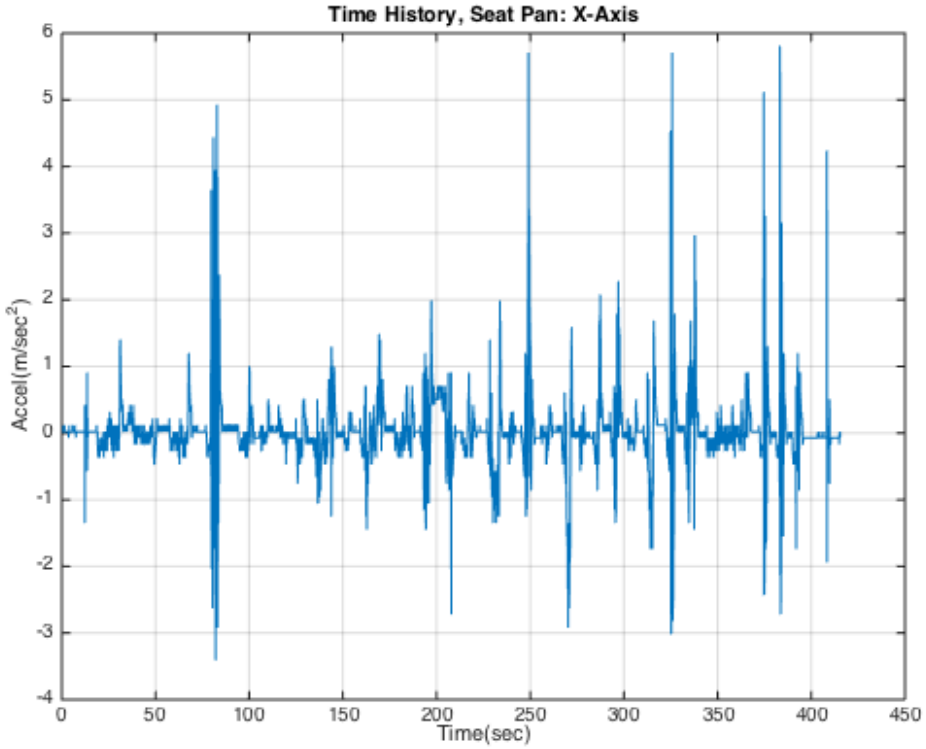

Time History, Back Plate: Y-Axis
Accel(m/sec^2)
Time(sec)
30
20
10
0
-10
-20
-30
0
50
100
150
200
250
300
350
400
450

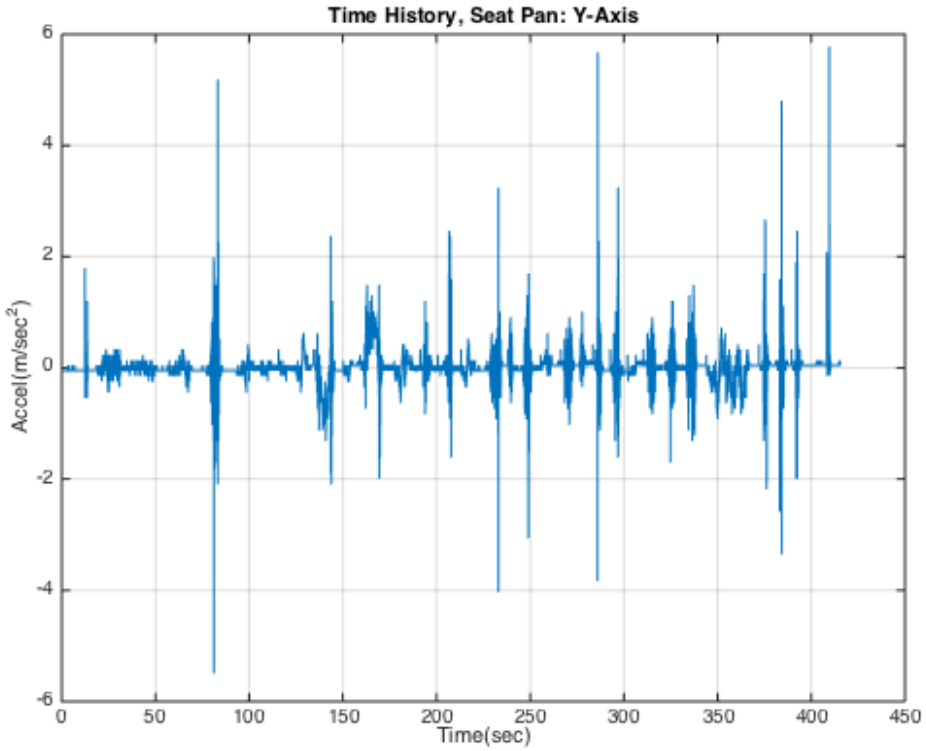

Time History, Back Plate: Z-Axis
Accel(m/sec^2)
Time(sec)
8
6
4
2
0
-2
-4
-6
-8
-10
-12
0
50
100
150
200
250
300
350
400
450

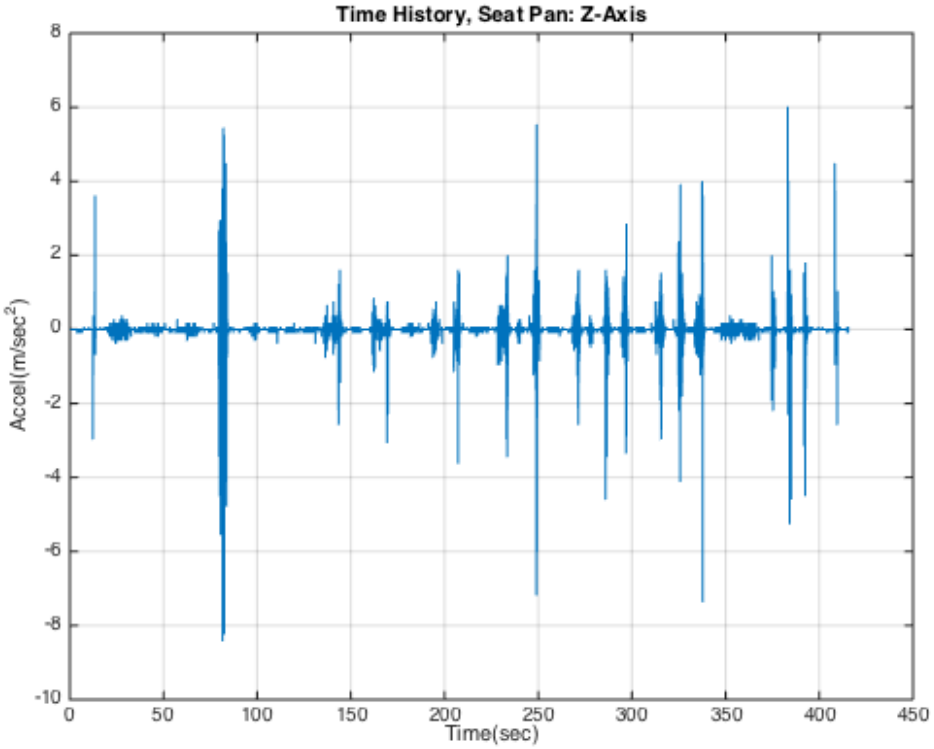

Time History, Chassis Frame: X-Axis
Accel(m/sec^2)
Time(sec)
6
4
2
0
-2
-4
-6
0
50
100
150
200
250
300
350
400
450


Time History, Chassis Frame: Y-Axis
Accel(m/sec^2)
Time(sec)
8
6
4
2
0
-2
-4
-6
-8
0
50
100
150
200
250
300
350
400
450


Time History, Chassis Frame: Z-Axis
Accel(m/sec^2)
Time(sec)
10
5
0
-5
-10
-15
0
50
100
150
200
250
300
350
400
450


Time History, Seat Pan: X-Axis
Accel(m/sec^2)
Time(sec)
6
5
4
3
2
1
0
-1
-2
-3
-4
0
50
100
150
200
250
300
350
400
450


Time History, Seat Pan: Y-Axis
Accel(m/sec^2)
Time(sec)
6
4
2
0
-2
-4
-6
0
50
100
150
200
250
300
350
400
450




```
%[fwl,fw,fwu,wk,wd,wf,wc,we,wj,wb,www,iweight]=weight_factors();
%
%
iweight = 0;
%
while( iweight ~= 1 && iweight ~= 2 && iweight ~= 3 && iweight ~= 4 && ...
        iweight ~= 5 && iweight ~= 6 && iweight ~= 7  )
    %
    %disp(' ');
    %disp(' Enter weighting by choice number: ');
    %
    %disp('  1= Wk  z-axis,      seat surface');
    %disp('  2= Wd  x or y-axis, seat surface');
    %disp('  3= Wf');
    %disp('  4= Wc');
    %disp('  5= We');
    %disp('  6= Wj');
    %disp('  7= Wb');
    %disp(' ');
    iweight=weighting(m);
```

```
    %
end
%
fc=zeros(44,1);
wk=zeros(44,1);
wd=zeros(44,1);
wf=zeros(44,1);
wc=zeros(44,1);
we=zeros(44,1);
wj=zeros(44,1);
wb=zeros(44,1);
%
%
fc(1)=0.02;
wk(1)=-90.;
wd(1)=-90.;
wf(1)=-32.33;
wc(1)=-90.;
we(1)=-90.;
wj(1)=-90.;
wb(1)=-90.;
%
fc(2)=0.025;
wk(2)=-90.;
wd(2)=-90.;
wf(2)=-28.48;
wc(2)=-90.;
we(2)=-90.;
wj(2)=-90.;
wb(2)=-90.;
%
fc(3)=0.0315;
wk(3)=-90.;
wd(3)=-90.;
wf(3)=-24.47;
wc(3)=-90.;
we(3)=-90.;
wj(3)= -90.;
```

```
wb(3)=-90.;
%
fc(4)=0.04;
wk(4)=-90.;
wd(4)=-90.;
wf(4)=-20.25;
wc(4)=-90.;
we(4)=-90.;
wj(4)=-90.;
wb(4)=-90.;
%
fc(5)=0.05;
wk(5)=-90.;
wd(5)=-90.;
wf(5)=-16.10;
wc(5)=-90.;
we(5)=-90.;
wj(5)=-90.;
wb(5)=-90.;
%
fc(6)=0.063;
wk(6)=-90.;
wd(6)=-90.;
wf(6)=-11.49;
wc(6)=-90.;
we(6)= -90.;
wj(6)=-90.;
wb(6)=-90.;
%
fc(7)=0.08;
wk(7)=-90.;
wd(7)=-90.;
wf(7)=-6.73;
wc(7)= -90.;
we(7)=-90.;
wj(7)=-90.;
wb(7)=-90.;
%
```

```
fc(8)=0.1;
wk(8)=-30.11;
wd(8)=-24.09;
wf(8)=-3.16;
wc(8)=-24.11;
we(8)=-24.08;
wj(8)=-30.18;
wb(8)=-32.04;
%
fc(9)=0.125;
wk(9)=-26.26;
wd(9)=-20.24;
wf(9)=-0.96;
wc(9)=-20.25;
we(9)=-20.22;
wj(9)=-26.32;
wb(9)=-28.20;
%
fc(10)=0.16;
wk(10)=-22.05;
wd(10)=-16.01;
wf(10)=0.05;
wc(10)=-16.03;
we(10)=-15.98;
wj(10)=-22.11;
wb(10)=-23.98;
%
fc(11)=0.2;
wk(11)=-18.33;
wd(11)=-12.28;
wf(11)=-0.07;
wc(11)=-12.03;
we(11)=-12.23;
wj(11)=-18.38;
wb(11)=-20.23;
%
fc(12)=0.25;
wk(12)=-14.81;
```

```
wd(12)=-8.75;
wf(12)=-1.37;
wc(12)=-8.78;
we(12)=-8.67;
wj(12)=-14.86;
wb(12)=-16.71;
%
fc(13)=0.315;
wk(13)=-11.60;
wd(13)=-5.52;
wf(13)=-4.17;
wc(13)=-5.56;
we(13)=-5.41;
wj(13)=-11.65;
wb(13)=-13.51;
%
fc(14)=0.4;
wk(14)=-9.07;
wd(14)=-2.94;
wf(14)=-8.31;
wc(14)=-3.01;
we(14)=-3.01;
wj(14)=-9.10;
wb(14)=-10.98;
%
fc(15)=0.5;
wk(15)=-7.57;
wd(15)=-1.38;
wf(15)=-13.00;
wc(15)=-1.48;
we(15)=-1.29;
wj(15)=-7.60;
wb(15)=-9.53;
%
fc(16)=0.63;
wk(16)=-6.77;
wd(16)=-0.50;
wf(16)=-18.69;
```

```
wc(16)=-0.64;
we(16)=-0.55;
wj(16)=-6.78;
wb(16)=-8.71;
%
fc(17)=0.8;
wk(17)=-6.43;
wd(17)=-0.07;
wf(17)=-25.51;
wc(17)=-0.24;
we(17)=-0.53;
wj(17)=-6.42;
wb(17)=-8.38;
%
fc(18)=1.;
wk(18)=-6.33;
wd(18)=0.10;
wf(18)=-32.57;
wc(18)=-0.08;
we(18)=-1.11;
wj(18)=-6.30;
wb(18)=-8.29;
%
fc(19)=1.25;
wk(19)=-6.29;
wd(19)=0.07;
wf(19)=-40.02;
wc(19)=0.00;
we(19)=-2.25;
wj(19)=-6.28;
wb(19)=-8.27;
%
fc(20)=1.6;
wk(20)=-6.12;
wd(20)=-0.28;
wf(20)=-48.47;
wc(20)=0.06;
we(20)=-3.99;
```

```
wj(20)=-6.32;
wb(20)=-8.07;
%
fc(21)=2.;
wk(21)=-5.49;
wd(21)=-1.01;
wf(21)=-56.19;
wc(21)=0.10;
we(21)=-5.82;
wj(21)=-6.34;
wb(21)=-7.60;
%
fc(22)=2.5;
wk(22)=-4.01;
wd(22)=-2.20;
wf(22)=-63.93;
wc(22)=0.15;
we(22)=-7.77;
wj(22)=-6.22;
wb(22)=-6.13;
%
fc(23)=3.15;
wk(23)=-1.90;
wd(23)=-3.85;
wf(23)=-71.96;
wc(23)=0.19;
we(23)=-9.81;
wj(23)=-5.62;
wb(23)=-3.58;
%
fc(24)=4.;
wk(24)=-0.29;
wd(24)=-5.82;
wf(24)=-80.26;
wc(24)=0.20;
we(24)=-11.93;
wj(24)=-4.04;
wb(24)=-1.02;
```

```
%
fc(25)=5.;
wk(25)=0.33;
wd(25)=-7.76;
wf(25)=-90.;
wc(25)=0.11;
we(25)=-13.91;
wj(25)=-2.01;
wb(25)=0.21;
%
fc(26)=6.3;
wk(26)=0.46;
wd(26)=-9.81;
wf(26)=-90.;
wc(26)=-0.23;
we(26)=-15.94;
wj(26)=-0.48;
wb(26)=0.46;
%
fc(27)=8.;
wk(27)=0.31;
wd(27)=-11.93;
wf(27)=-90.;
wc(27)=-1.00;
we(27)=-18.03;
wj(27)=0.15;
wb(27)=0.21;
%
fc(28)=10.;
wk(28)=-0.10;
wd(28)=-13.91;
wf(28)=-90.;
wc(28)=-2.20;
we(28)=-19.98;
wj(28)=0.26;
wb(28)=-0.23;
%
fc(29)=12.5;
```

```
wk(29)=-0.89;
wd(29)=-15.87;
wf(29)=-90.;
wc(29)=-3.79;
we(29)=-21.93;
wj(29)=0.22;
wb(29)=-0.85;
%
fc(30)=16.;
wk(30)=-2.28;
wd(30)=-18.03;
wf(30)=-90.;
wc(30)=-5.82;
we(30)=-24.08;
wj(30)=0.16;
wb(30)=-1.83;
%
fc(31)=20.;
wk(31)=-3.93;
wd(31)=-19.99;
wf(31)=-90.;
wc(31)=-7.77;
we(31)=-26.02;
wj(31)=0.10;
wb(31)=-3.00;
%
fc(32)=25.;
wk(32)=-5.80;
wd(32)=-21.94;
wf(32)=-90.;
wc(32)=-9.76;
we(32)=-27.97;
wj(32)=0.06;
wb(32)=-4.44;
%
fc(33)=31.5;
wk(33)=-7.86;
wd(33)=-23.98;
```

```
wf(33)=-90.;
wc(33)=-11.84;
we(33)=-30.01;
wj(33)=0.00;
wb(33)=-6.16;
%
fc(34)=40.;
wk(34)=-10.05;
wd(34)=-26.13;
wf(34)=-90.;
wc(34)=-14.02;
we(34)=-32.15;
wj(34)=-0.08;
wb(34)=-8.11;
%
fc(35)=50.;
wk(35)=-12.19;
wd(35)=-28.22;
wf(35)=-90.;
wc(35)=-16.13;
we(35)=-34.24;
wj(35)=-0.24;
wb(35)=-10.09;
%
fc(36)=63.;
wk(36)=-14.61;
wd(36)=-30.60;
wf(36)=-90.;
wc(36)=-18.53;
we(36)=-36.62;
wj(36)=-0.62;
wb(36)=-12.43;
%
fc(37)=80.;
wk(37)=-17.56;
wd(37)=-33.53;
wf(37)=-90.;
wc(37)=-21.47;
```

```
        we(37)=-39.55;
        wj(37)=-1.48;
        wb(37)=-15.34;
        %
        fc(38)=100.;
        wk(38)=-21.04;
        wd(38)=-36.99;
        wf(38)=-90.;
        wc(38)=-24.94;
        we(38)=-43.01;
        wj(38)=-3.01;
        wb(38)=-18.72;
        %
        fc(39)=125.;
        wk(39)=-25.35;
        wd(39)=-41.28;
        wf(39)=-90.;
        wc(39)=-29.24;
        we(39)=-47.31;
        wj(39)=-5.36;
        wb(39)=-23.00;
        %
        fc(40)=160.;
        wk(40)=-30.91;
        wd(40)=-46.84;
        wf(40)=-90.;
        wc(40)=-34.80;
        we(40)=-52.86;
        wj(40)=-8.78;
        wb(40)=-28.56;
        %
        fc(41)=200.;
        wk(41)=-36.38;
        wd(41)=-52.30;
        wf(41)=-90.;
        wc(41)=-40.26;
        we(41)=-58.33;
        wj(41)=-12.30;
```

```
        wb(41)=-34.03;
        %
        fc(42)=250.;
        wk(42)=-42.0;
        wd(42)=-57.97;
        wf(42)=-90.;
        wc(42)=-45.92;
        we(42)=-63.99;
        wj(42)=-16.03;
        wb(42)=-39.69;
        %
        fc(43)=315.;
        wk(43)=-48.00;
        wd(43)=-63.92;
        wf(43)=-90.;
        wc(43)=-51.88;
        we(43)=-69.94;
        wj(43)=-19.98;
        wb(43)=-46.65;
        %
        fc(44)=400.;
        wk(44)=-54.20;
        wd(44)=-70.12;
        wf(44)=-90.;
        wc(44)=-58.08;
        we(44)=-76.14;
        wj(44)=-24.10;
        wb(44)=-51.84;
        %
        for i=1:44
            %
            wk(i)=10^(wk(i)/20.);
            wd(i)=10^(wd(i)/20.);
            wf(i)=10^(wf(i)/20.);
            wc(i)=10^(wc(i)/20.);
            we(i)=10^(we(i)/20.);
            wj(i)=10^(wj(i)/20.);
            wb(i)=10^(wb(i)/20.);
```

```
    %
end
%
www=zeros(44,1);
%
for i=1:44
    %
    if(iweight==1)
        www(i)=wk(i);
    end
    if(iweight==2)
        www(i)=wd(i);
    end
    if(iweight==3)
        www(i)=wf(i);
    end
    if(iweight==4)
        www(i)=wc(i);
    end
    if(iweight==5)
        www(i)=we(i);
    end
    if(iweight==6)
        www(i)=wj(i);
    end
    if(iweight==7)
        www(i)=wb(i);
    end
    %
end
%
imax=length(fc);
%
fw=fc;
%
fwl=fw/(2^(1/6));
%
fwu=zeros(imax,1);
```

```
%
for i=1:imax-1
    fwu(i)=fwl(i+1);
end
%
fwu(imax)=fw(imax)*(2^(1/6));
aw=zeros(length(amp),1);
%
iband=3;  % bandpass filtering
iphase=1; % refiltering for phase correction
%
progressbar;
for i=1:44
    %
    progressbar(i/44);
    %
    fh=fwl(i);  % highpass filter frequency
    fl=fwu(i);  % lowpass filter frequency
    %
    if(fl<sr/2.1)
        [y,mu,sd,rms(i)]=...
            Butterworth_filter_function_alt(amp,dt,iband,fl,fh,iphase);
        %
        aw=aw+y*www(i);
    end
    %
end
%
pause(0.5);
progressbar(1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
n=length(aw);
%
AW=std(aw);
AU=std(amp);
MTW=AW;
MTU=AU;
```

```
%
VDVu=0;
VDVu_run=zeros(n,1);
for i=1:n
    VDVu=VDVu+amp(i)^4;
    VDVu_run(i,1)=(VDVu*dt)^(0.25);
end
VDVu=(VDVu*dt)^0.25;
%VDVu_run=transpose(VDVu_run);
%
VDVw=0;
VDVw_run=zeros(n,1);
for i=1:n
    VDVw=VDVw+aw(i)^4;
    VDVw_run(i,1)=(VDVw*dt)^0.25;
end
VDVw=(VDVw*dt)^0.25;
%VDVw_run=transpose(VDVw_run);
%
aurms=0;
aurms_run=zeros(n,1);
for i=1:n
    aurms=aurms+amp(i)^2;
    aurms_run(i,1)=(aurms*dt/tim(i))^0.5;
end
aurms=aurms_run(n);
%aurms_run=transpose(aurms_run);
%
awrms=0;
awrms_run=zeros(n,1);
for i=1:n
    awrms=awrms+aw(i)^2;
    awrms_run(i,1)=(awrms*dt/tim(i))^0.5;
end
awrms=awrms_run(n);
%awrms_run=awrms_run.';
%
%disp(' ');
```

```
%disp(' Subdivide time history into segments?  1=yes 2=no');
%ig=input(' ');
ig=1;
%
if(ig==1)
    %disp(' ');
    %dur=input(' Enter segment duration (sec) ');
    dur=1;
    %
    ns=round(dur/dt);
    %
    loops=floor(n/ns);
    %
    ia=1;
    %
    disp(' ');
    %disp(' Time          aw     ');
    %disp(' (sec)     (m/sec^2) RMS ');
    %
    for i=1:loops
        ib=ia+ns-1;
        if(ib>n)
            break;
        end
        ttt=dt*(ib+ia)/2;
        ttt_run(i)=ttt;
        asu=std(amp(ia:ib));
        asu_run(i)=asu;
        out1=sprintf('%8.0f  %8.2f ',ttt,asu);
        %disp(out1);
        if(asu>MTU)
            MTU=asu;
        end
        ia=ib;
    end
    %
    ia=1;
    %
```

```
            for i=1:loops
                ib=ia+ns-1;
                if(ib>n)
                    break;
                end
                ttt=dt*(ib+ia)/2;
                ttt_run(i)=ttt;
                asw=std(aw(ia:ib));
                asw_run(i)=asw;
                out1=sprintf('%8.0f  %8.2f ',ttt,asw);
                %disp(out1);
                if(asw>MTW)
                    MTW=asw;
                end
                ia=ib;
            end
            %
        end
        disp(' ');
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %
        disp(name{m})

        %
        out1=sprintf('\n Unweighted R.M.S. Acceleration  aurms = %8.4g (m/sec^2)RMS
',aurms);
        disp(out1);
        %
        out1=sprintf('\n Unweighted Fourth Power Vibration Dose VDVu = %8.4g (m/
sec^(1.75)) \n',VDVu);
        disp(out1);
        %
        %
        out1=sprintf('\n Peak Weighted Acceleration  max(aw(t)) = %8.4g m/sec^2
',max(aw));
        disp(out1);
        %
```

```
        out1=sprintf('\n Weighted R.M.S. Acceleration  awrms = %8.4g (m/sec^2)RMS
',awrms);
        disp(out1);
        %
        out1=sprintf('\n Weighted Fourth Power Vibration Dose VDVw = %8.4g (m/
sec^(1.75)) \n',VDVw);
        disp(out1);
        %
        out1=sprintf('\n                          CF = %8.4g\n',max(aw)/awrms);
        disp(out1);
```

```
Back Plate: X-Axis

 Unweighted R.M.S. Acceleration  aurms =    1.274 (m/sec^2)RMS

 Unweighted Fourth Power Vibration Dose VDVu =    20.54 (m/sec^(1.75))


 Peak Weighted Acceleration  max(aw(t)) =    35.83 m/sec^2

 Weighted R.M.S. Acceleration  awrms =    1.218 (m/sec^2)RMS

 Weighted Fourth Power Vibration Dose VDVw =    20.34 (m/sec^(1.75))


                         CF =    29.42

Back Plate: Y-Axis

 Unweighted R.M.S. Acceleration  aurms =    0.764 (m/sec^2)RMS

 Unweighted Fourth Power Vibration Dose VDVu =       14 (m/sec^(1.75))


 Peak Weighted Acceleration  max(aw(t)) =    5.741 m/sec^2

 Weighted R.M.S. Acceleration  awrms =   0.3367 (m/sec^2)RMS

 Weighted Fourth Power Vibration Dose VDVw =    4.363 (m/sec^(1.75))
```

CF = 17.05

Back Plate: Z-Axis

Unweighted R.M.S. Acceleration aurms = 0.5219 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 7.811 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 4.47 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.2857 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 3.776 (m/sec^(1.75))

CF = 15.65

Chassis Frame: X-Axis

Unweighted R.M.S. Acceleration aurms = 0.4685 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 4.678 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 4.175 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.2752 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 3.251 (m/sec^(1.75))

CF = 15.17

Chassis Frame: Y-Axis

Unweighted R.M.S. Acceleration aurms = 0.3604 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 4.64 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 2.588 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.1872 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 2.058 (m/sec^(1.75))

CF = 13.82

Chassis Frame: Z-Axis

Unweighted R.M.S. Acceleration aurms = 0.5362 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 8.516 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 7.079 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.4839 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 7.272 (m/sec^(1.75))

CF = 14.63

Seat Pan: X-Axis

Unweighted R.M.S. Acceleration aurms = 0.4789 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 4.857 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 4.365 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.3091 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 3.688 (m/sec^(1.75))

CF = 14.12

Seat Pan: Y-Axis

Unweighted R.M.S. Acceleration aurms = 0.2858 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 3.68 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 2.026 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.1265 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 1.578 (m/sec^(1.75))

CF = 16.02

Seat Pan: Z-Axis

Unweighted R.M.S. Acceleration aurms = 0.4046 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 5.999 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 5.925 m/sec^2

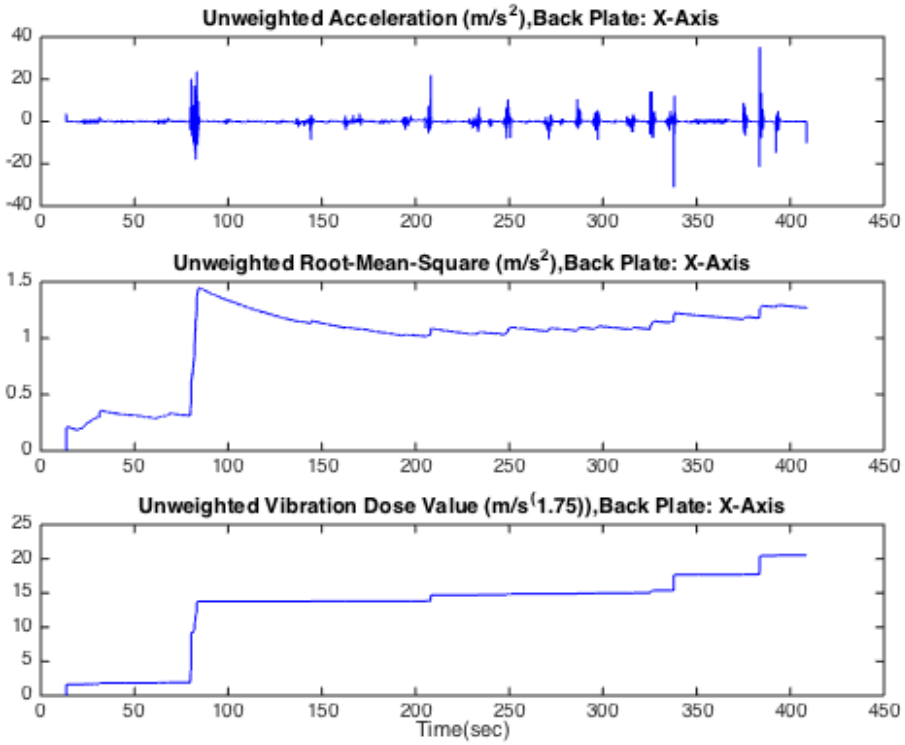
Weighted R.M.S. Acceleration awrms = 0.3707 (m/sec^2)RMS

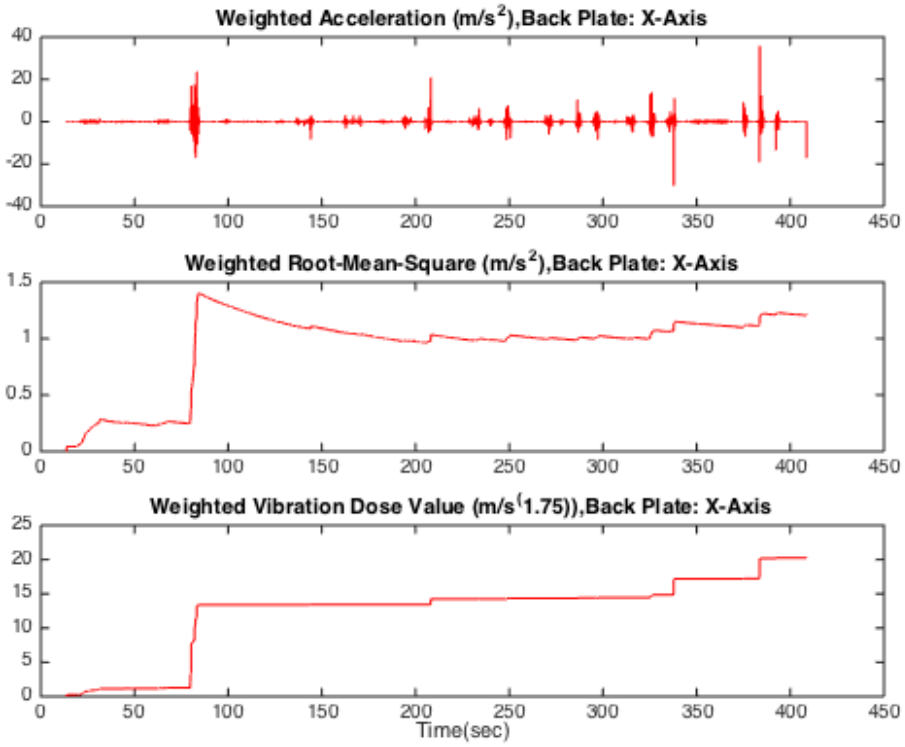
Weighted Fourth Power Vibration Dose VDVw = 5.382 (m/sec^(1.75))

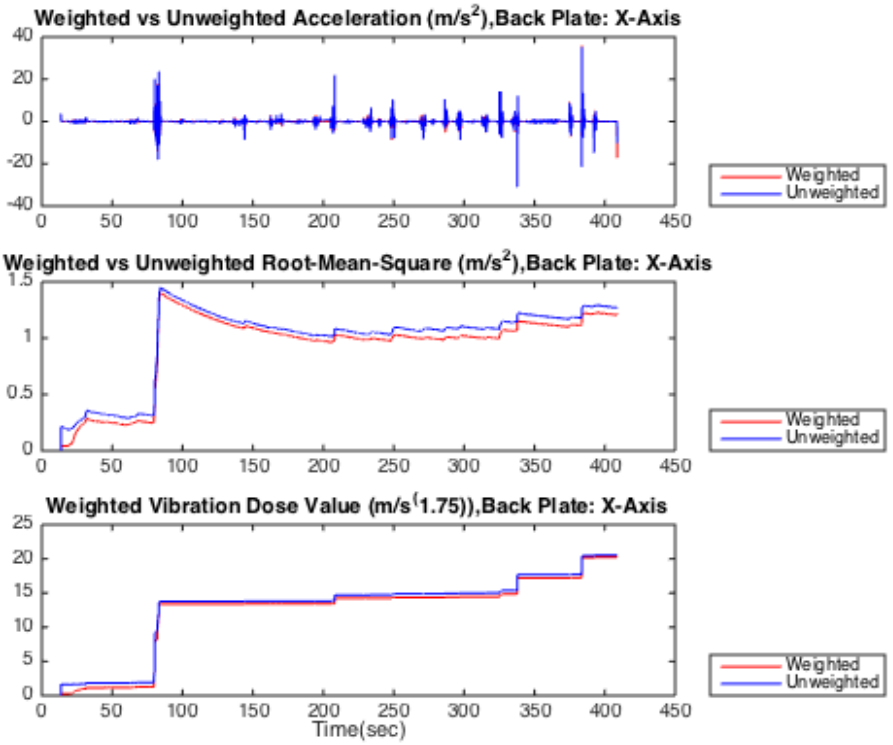
```
CF =    15.98
```

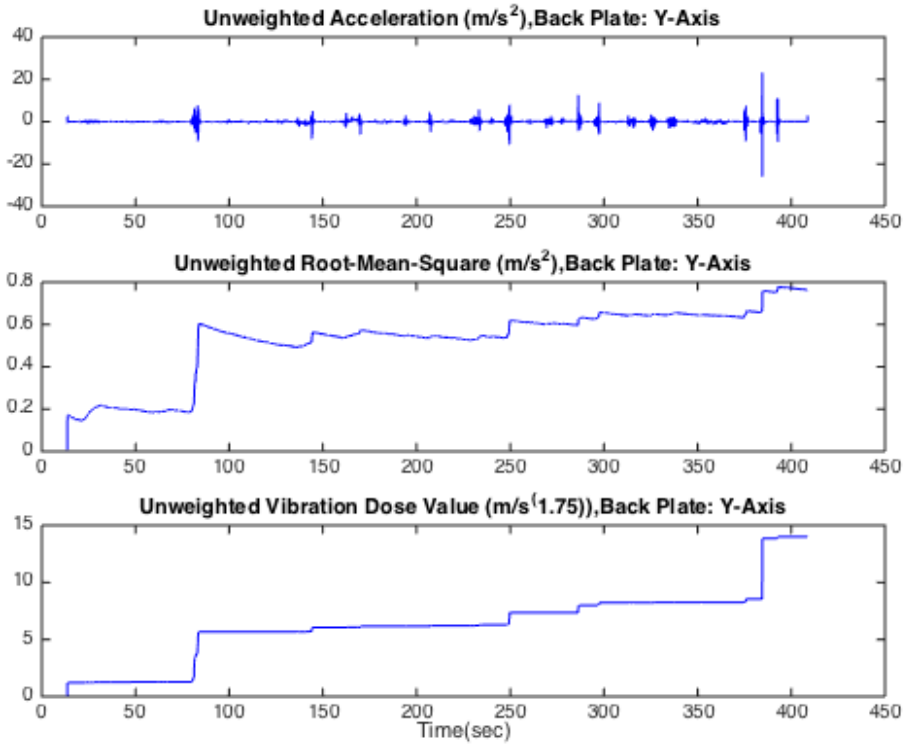
```
figure
title(['Unweighted Vibration Analysis,',name{m}])
subplot (3,1,1)
plot(tim,amp,'b')
title(['Unweighted Acceleration (m/s^2),',name{m}])

subplot (3,1,2)
plot(tim,aurms_run,'b')
title(['Unweighted Root-Mean-Square (m/s^2),',name{m}])

subplot (3,1,3)
plot(tim,VDVu_run,'b')
title(['Unweighted Vibration Dose Value (m/s^(1.75)),',name{m}])
xlabel('Time(sec)');
%
figure
title(['Weighted Vibration Analysis,',name{m}])
subplot (3,1,1)
plot(tim,aw,'r')
title(['Weighted Acceleration (m/s^2),',name{m}])

subplot (3,1,2)
plot(tim,awrms_run,'r')
title(['Weighted Root-Mean-Square (m/s^2),',name{m}])

subplot (3,1,3)
plot(tim,VDVw_run,'r')
title(['Weighted Vibration Dose Value (m/s^(1.75)),',name{m}])
xlabel('Time(sec)');
%
%
figure
title(['Weighted vs Unweighted Vibration Analysis,',name{m}])
subplot (3,1,1)
plot(tim,aw,'r',tim,amp,'b')
title(['Weighted vs Unweighted Acceleration (m/s^2),',name{m}])
```

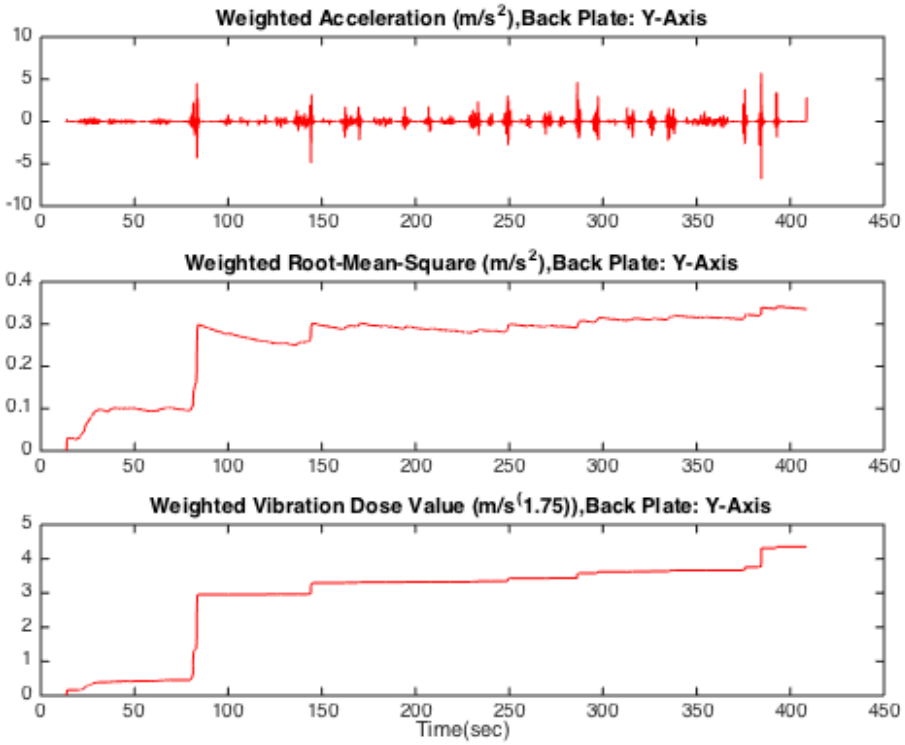
```
        legend('Weighted','Unweighted','Location','SouthEastOutside')

        subplot (3,1,2)
        plot(tim,awrms_run,'r',tim,aurms_run,'b')
        title(['Weighted vs Unweighted Root-Mean-Square (m/s^2),',name{m}])
        legend('Weighted','Unweighted','Location','SouthEastOutside')

        subplot (3,1,3)
        plot(tim,VDVw_run,'r',tim,VDVu_run,'b')
        title(['Weighted Vibration Dose Value (m/s^(1.75)),',name{m}])
        legend('Weighted','Unweighted','Location','SouthEastOutside')
        xlabel('Time(sec)');
```

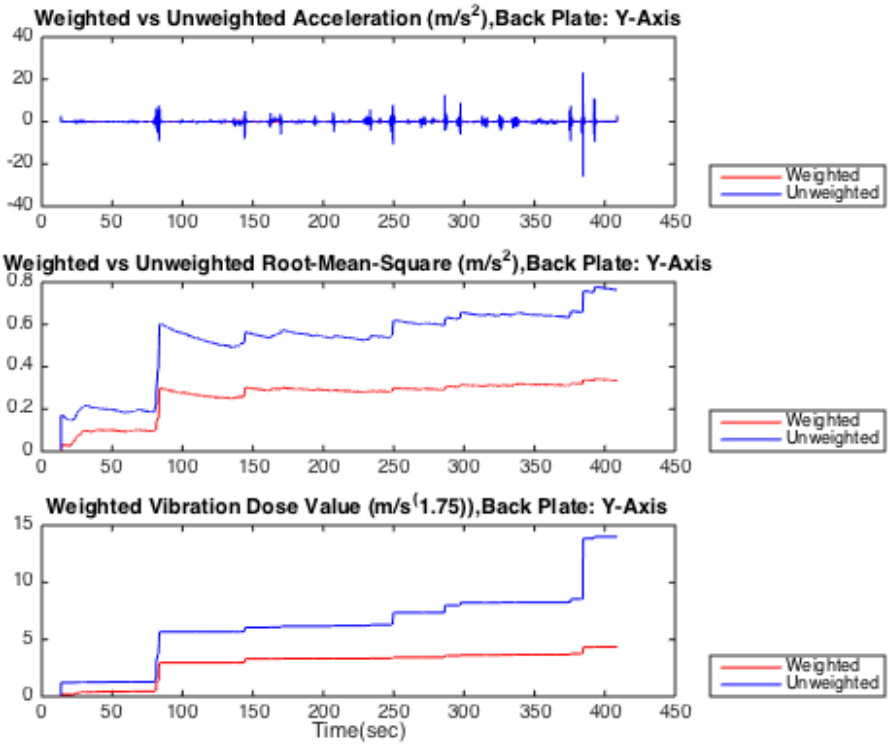

Unweighted Acceleration (m/s2),Back Plate: X-Axis
Unweighted Root-Mean-Square (m/s2),Back Plate: X-Axis
Unweighted Vibration Dose Value (m/s(1.75)),Back Plate: X-Axis
Time(sec)

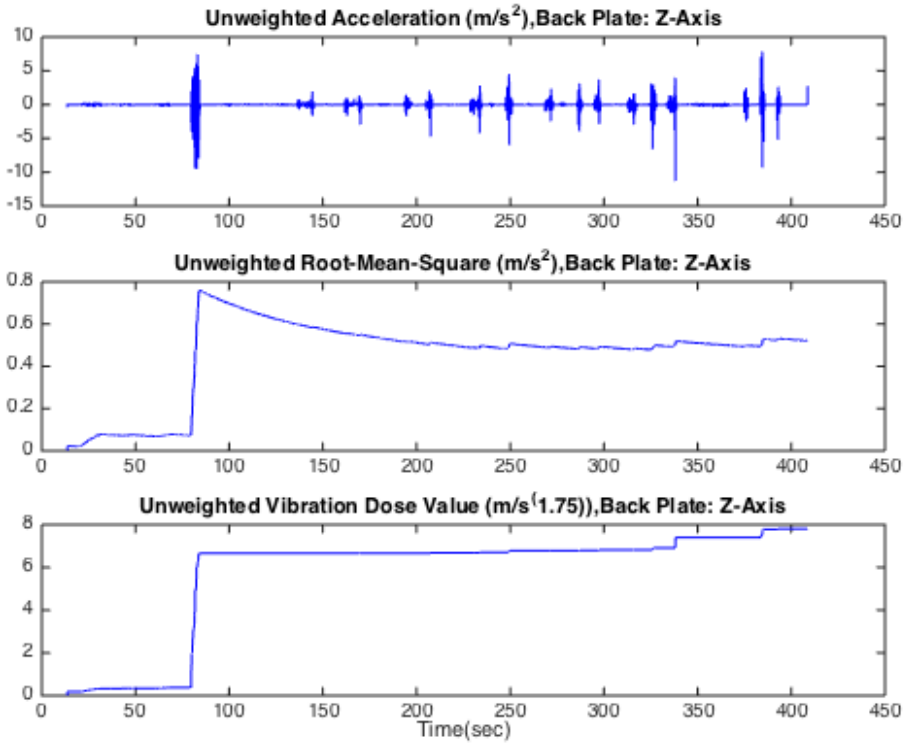

Weighted Acceleration (m/s2),Back Plate: X-Axis
Weighted Root-Mean-Square (m/s2),Back Plate: X-Axis
Weighted Vibration Dose Value (m/s(1.75)),Back Plate: X-Axis
Time(sec)

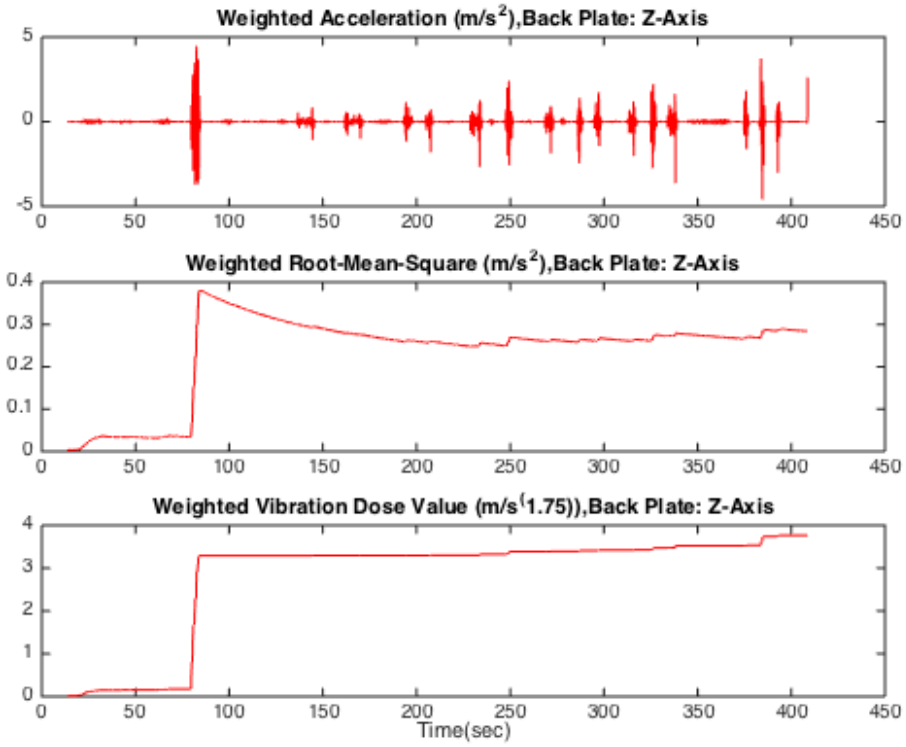

Weighted vs Unweighted Acceleration (m/s2),Back Plate: X-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s2),Back Plate: X-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s(1.75)),Back Plate: X-Axis
Weighted
Unweighted
Time(sec)

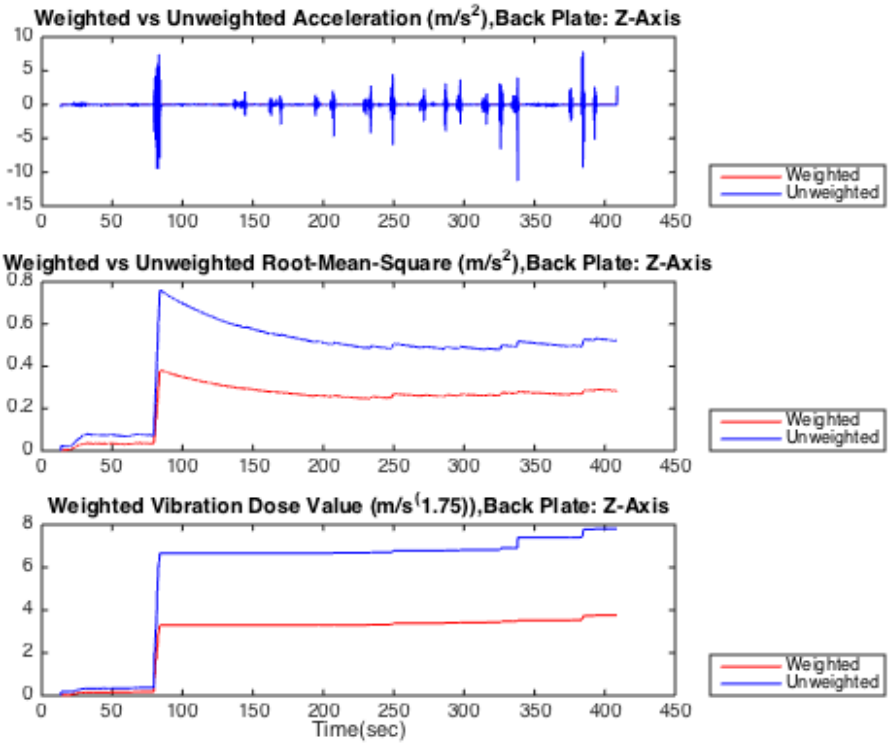

Unweighted Acceleration (m/s2),Back Plate: Y-Axis
Unweighted Root-Mean-Square (m/s2),Back Plate: Y-Axis
Unweighted Vibration Dose Value (m/s(1.75)),Back Plate: Y-Axis
Time(sec)

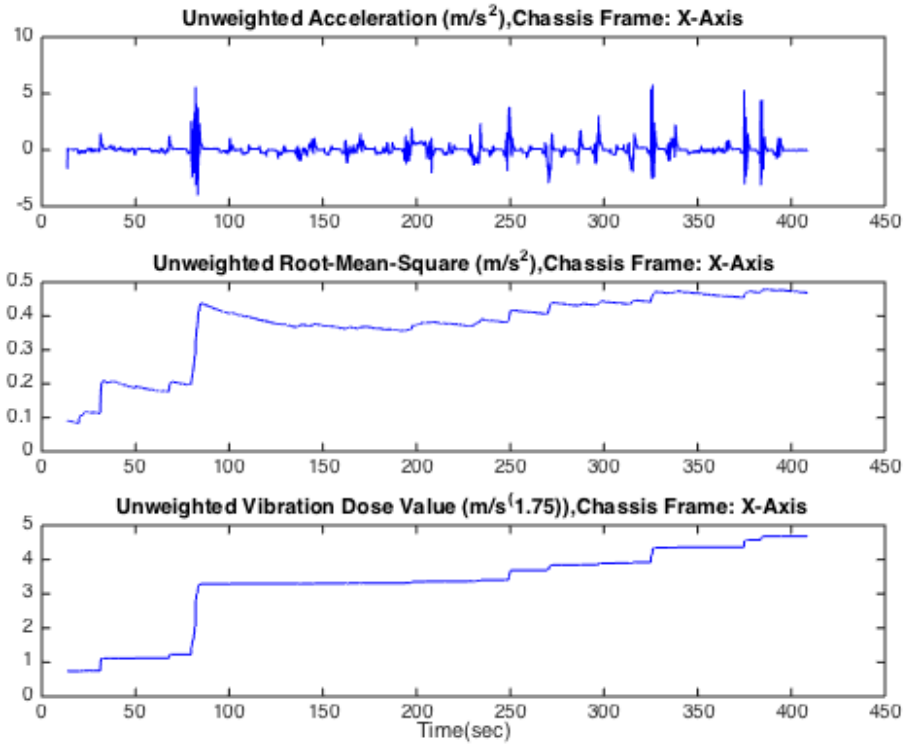

Weighted Acceleration (m/s2),Back Plate: Y-Axis
Weighted Root-Mean-Square (m/s2),Back Plate: Y-Axis
Weighted Vibration Dose Value (m/s1.75)),Back Plate: Y-Axis
Time(sec)

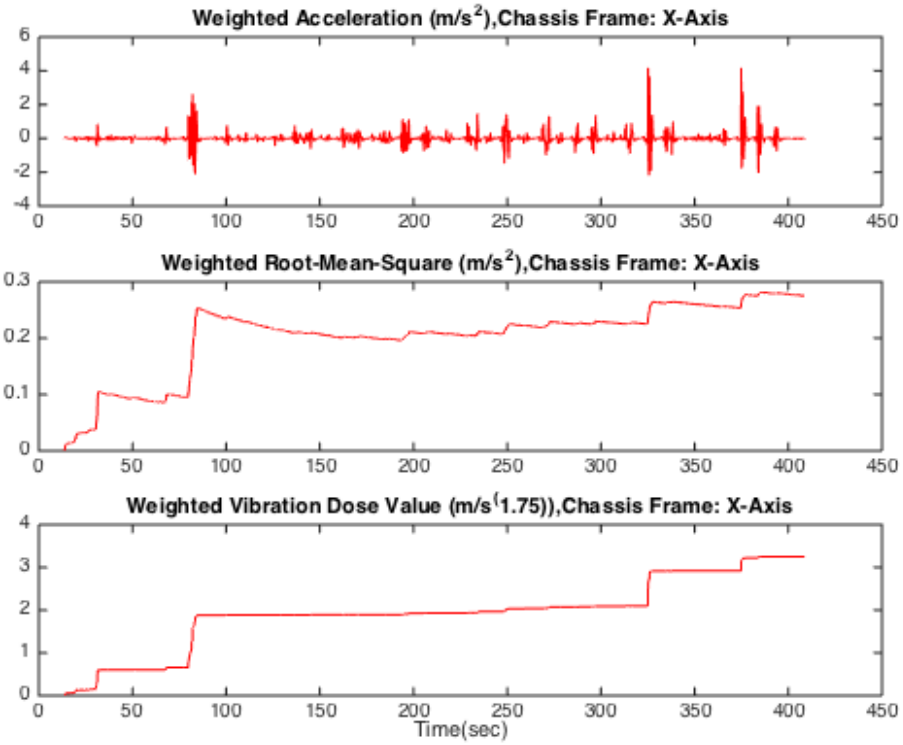

Weighted vs Unweighted Acceleration (m/s2),Back Plate: Y-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s2),Back Plate: Y-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s1.75)),Back Plate: Y-Axis
Weighted
Unweighted
Time(sec)

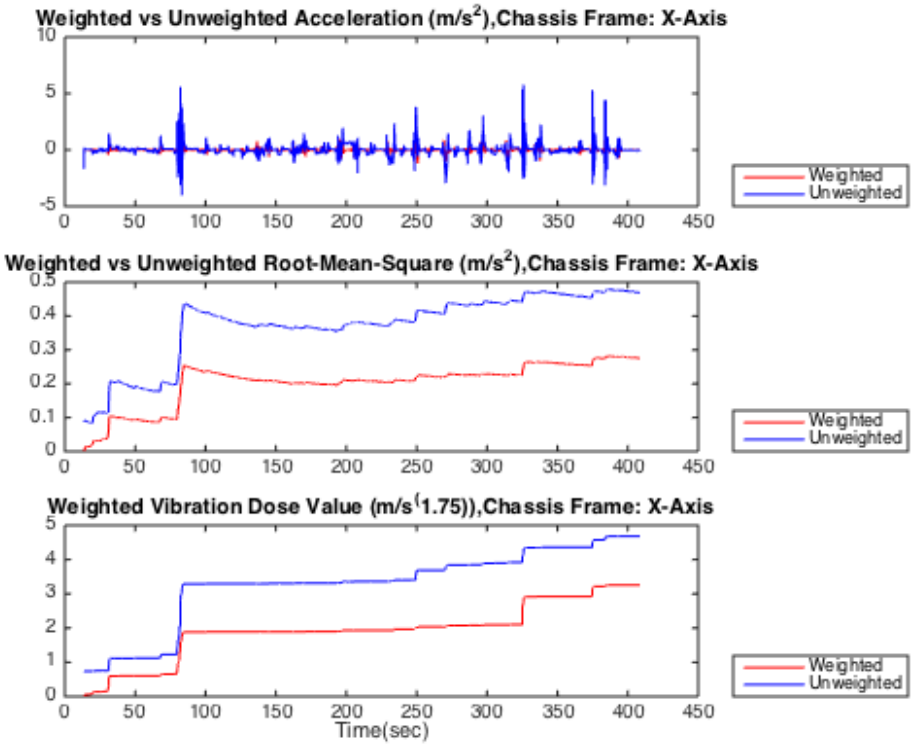

Unweighted Acceleration (m/s^2),Back Plate: Z-Axis
Unweighted Root-Mean-Square (m/s^2),Back Plate: Z-Axis
Unweighted Vibration Dose Value (m/s^1.75)),Back Plate: Z-Axis
Time(sec)

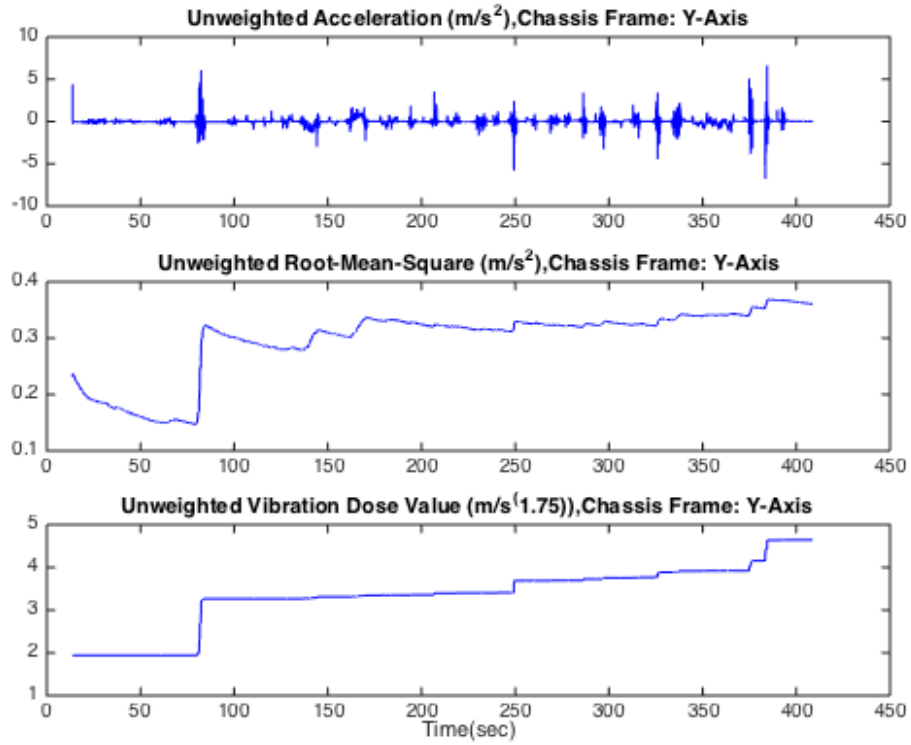

Weighted Acceleration (m/s^2),Back Plate: Z-Axis
Weighted Root-Mean-Square (m/s^2),Back Plate: Z-Axis
Weighted Vibration Dose Value (m/s^1.75)),Back Plate: Z-Axis
Time(sec)

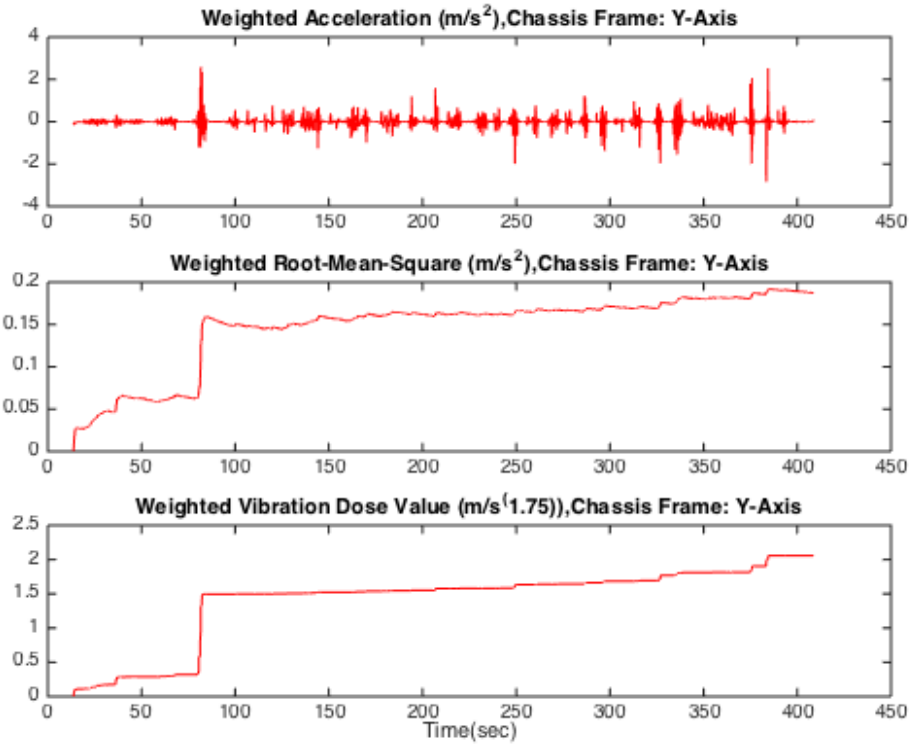

Weighted vs Unweighted Acceleration (m/s2),Back Plate: Z-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s2),Back Plate: Z-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s1.75)),Back Plate: Z-Axis
Weighted
Unweighted
Time(sec)

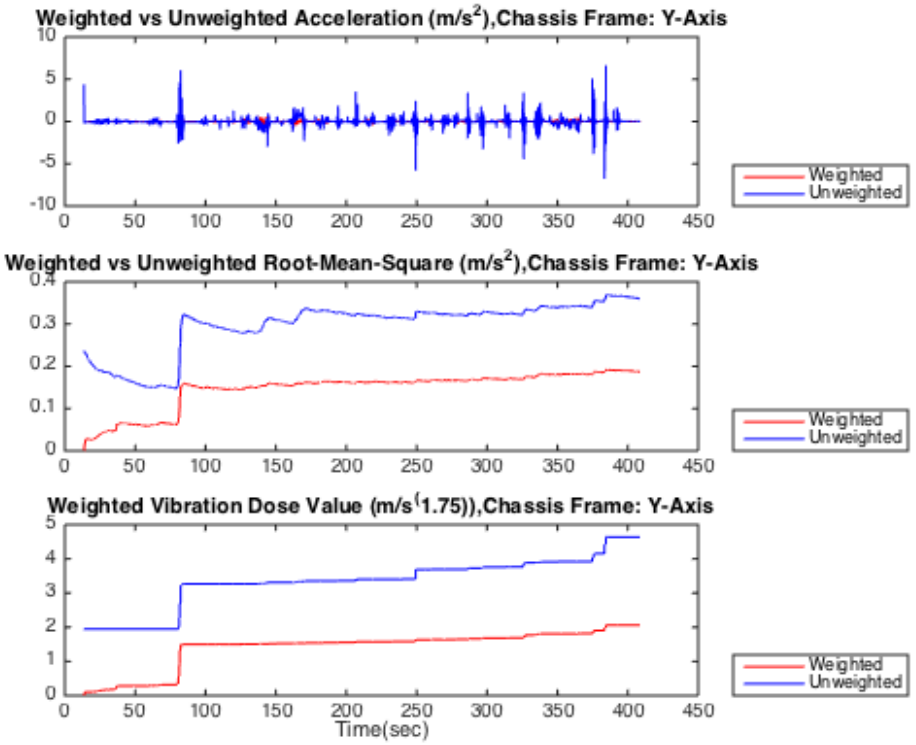

Unweighted Acceleration (m/s2),Chassis Frame: X-Axis
Unweighted Root-Mean-Square (m/s2),Chassis Frame: X-Axis
Unweighted Vibration Dose Value (m/s1.75)),Chassis Frame: X-Axis
Time(sec)


Weighted Acceleration (m/s2),Chassis Frame: X-Axis
Weighted Root-Mean-Square (m/s2),Chassis Frame: X-Axis
Weighted Vibration Dose Value (m/s(1.75)),Chassis Frame: X-Axis
Time(sec)


Weighted vs Unweighted Acceleration (m/s2),Chassis Frame: X-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s2),Chassis Frame: X-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s(1.75)),Chassis Frame: X-Axis
Weighted
Unweighted
Time(sec)

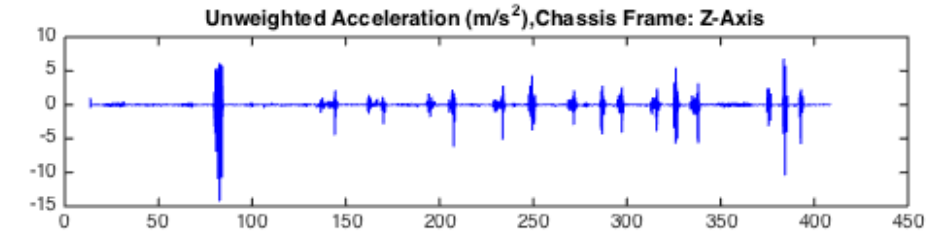

Unweighted Acceleration (m/s^2),Chassis Frame: Y-Axis
Unweighted Root-Mean-Square (m/s^2),Chassis Frame: Y-Axis
Unweighted Vibration Dose Value (m/s^1.75)),Chassis Frame: Y-Axis
Time(sec)

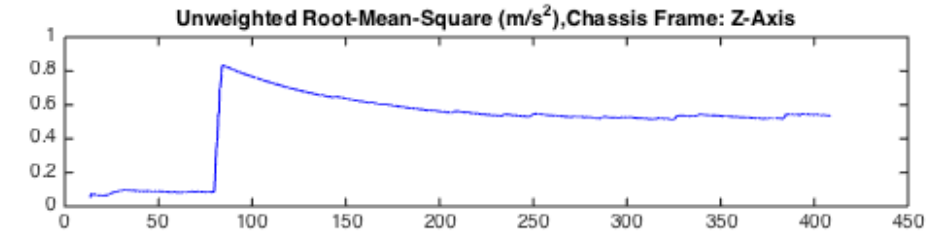

Weighted Acceleration (m/s^2),Chassis Frame: Y-Axis
Weighted Root-Mean-Square (m/s^2),Chassis Frame: Y-Axis
Weighted Vibration Dose Value (m/s^1.75)),Chassis Frame: Y-Axis
Time(sec)

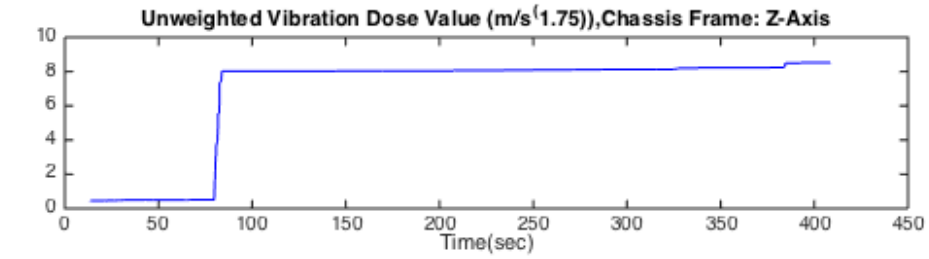

Weighted vs Unweighted Acceleration (m/s2),Chassis Frame: Y-Axis
Weighted
Unweighted

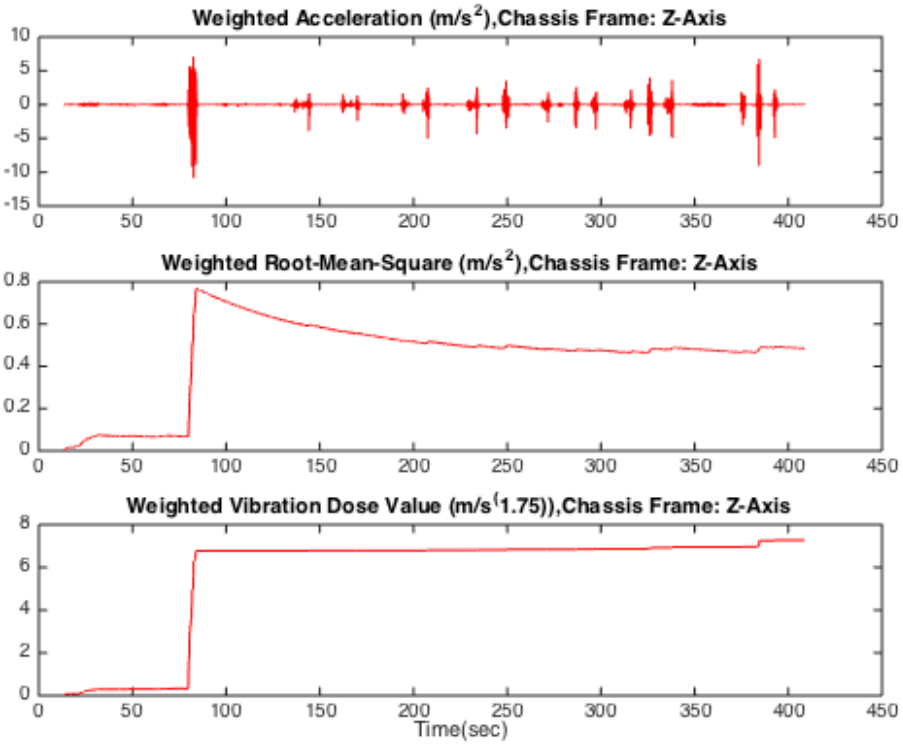
Weighted vs Unweighted Root-Mean-Square (m/s2),Chassis Frame: Y-Axis
Weighted
Unweighted

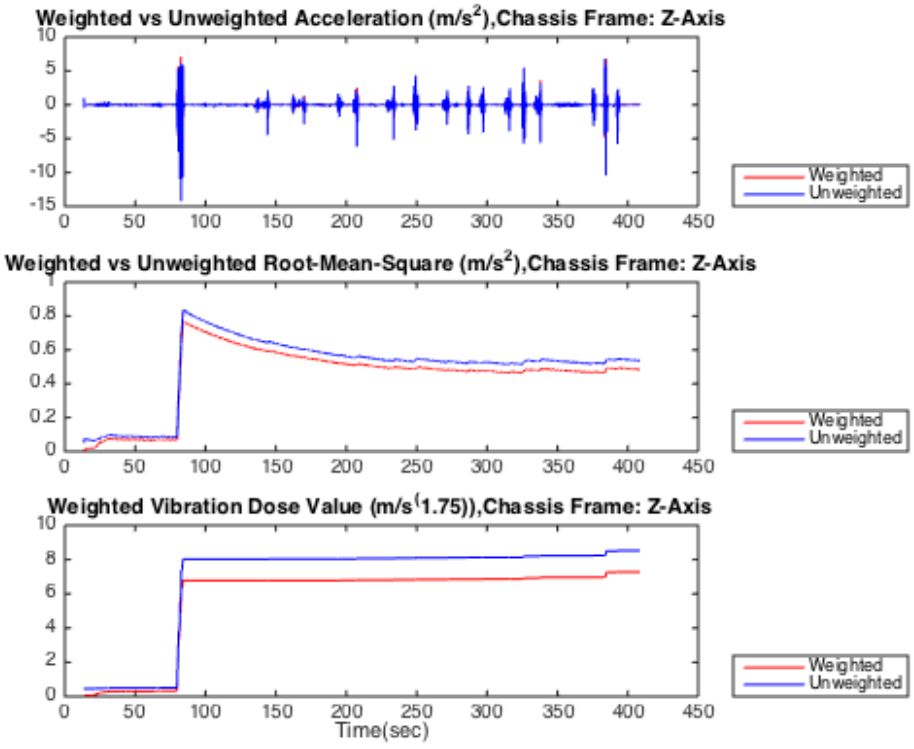
Weighted Vibration Dose Value (m/s1.75)),Chassis Frame: Y-Axis
Weighted
Unweighted
Time(sec)

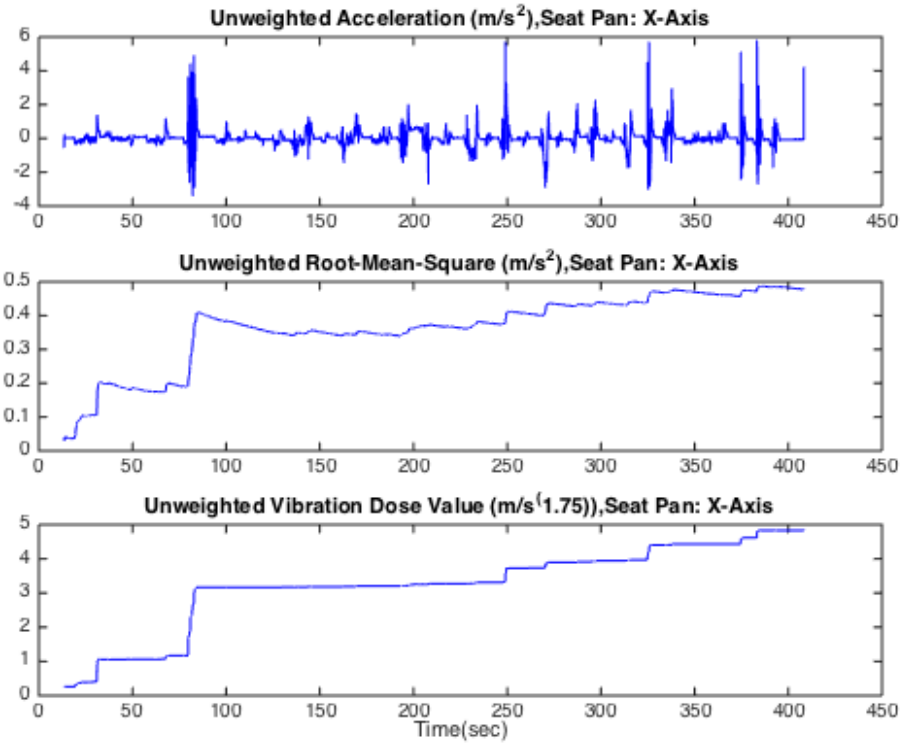

Unweighted Acceleration (m/s2),Chassis Frame: Z-Axis

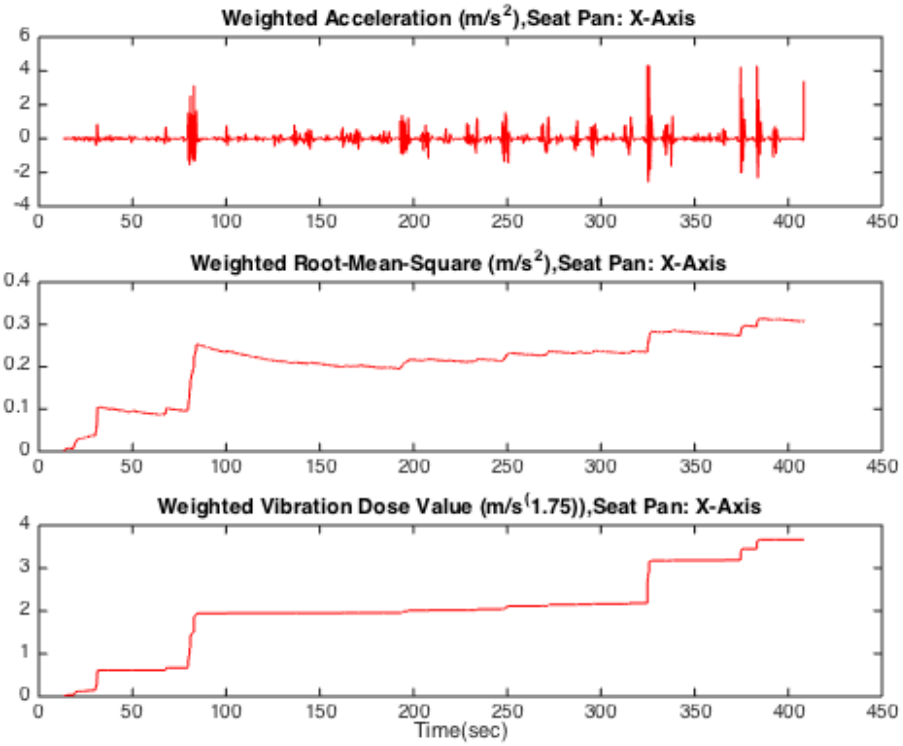

Unweighted Root-Mean-Square (m/s2),Chassis Frame: Z-Axis

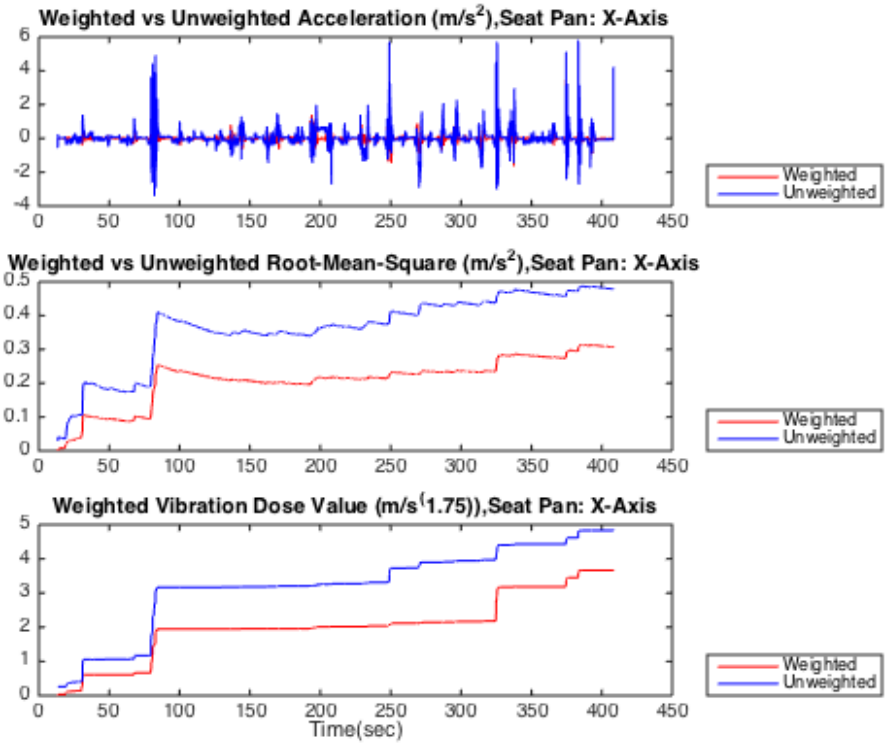

Unweighted Vibration Dose Value (m/s1.75)),Chassis Frame: Z-Axis
Time(sec)

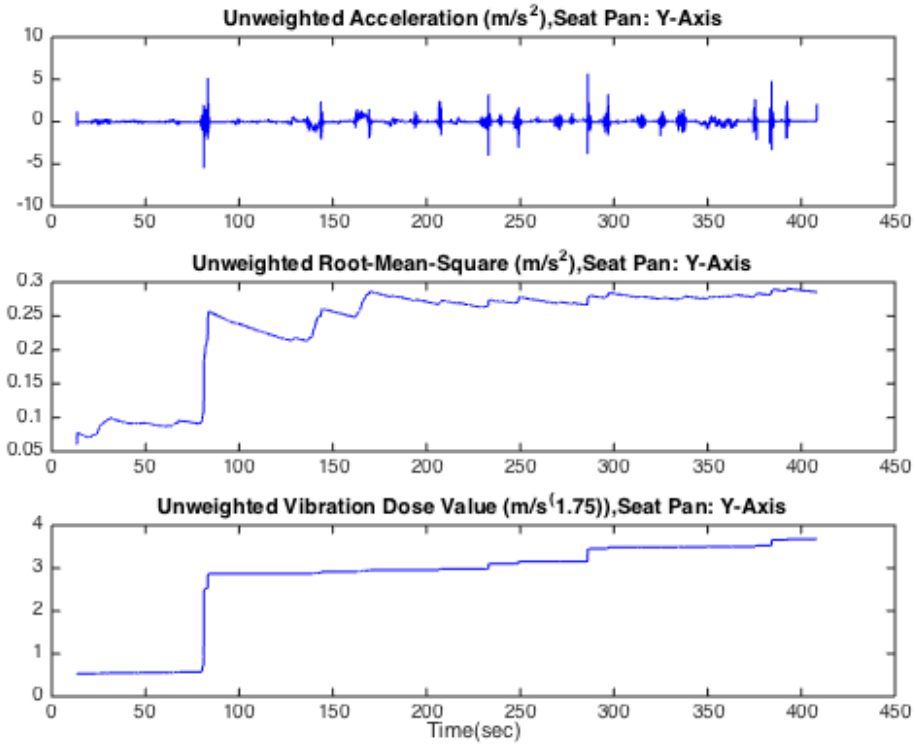

Weighted Acceleration (m/s^2),Chassis Frame: Z-Axis
Weighted Root-Mean-Square (m/s^2),Chassis Frame: Z-Axis
Weighted Vibration Dose Value (m/s$^{1.75}$),Chassis Frame: Z-Axis
Time(sec)

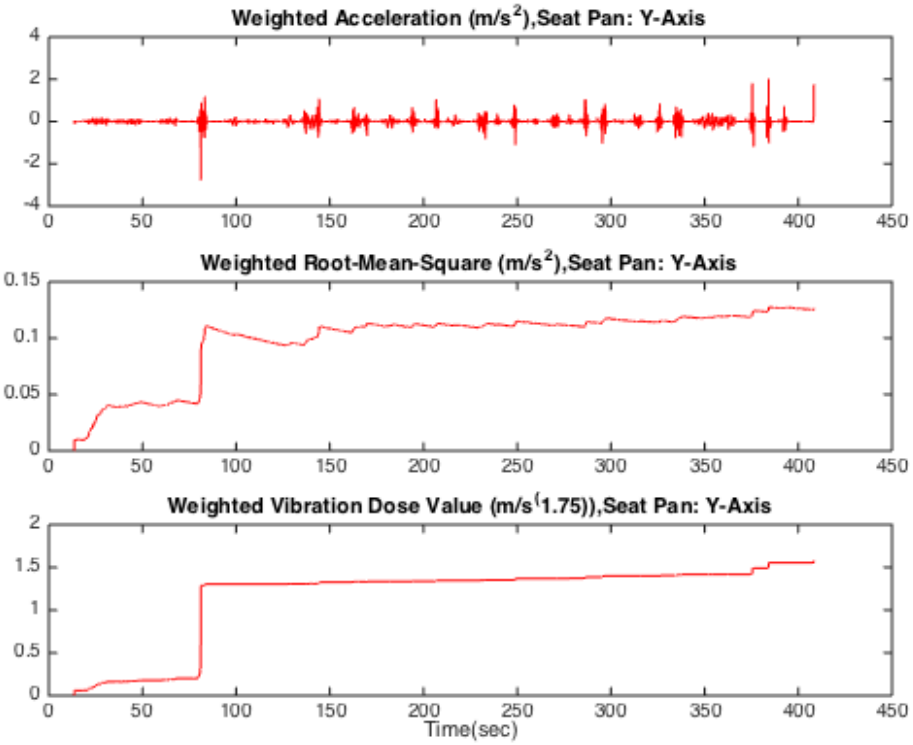

Weighted vs Unweighted Acceleration (m/s^2),Chassis Frame: Z-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s^2),Chassis Frame: Z-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s$^{1.75}$),Chassis Frame: Z-Axis
Weighted
Unweighted
Time(sec)

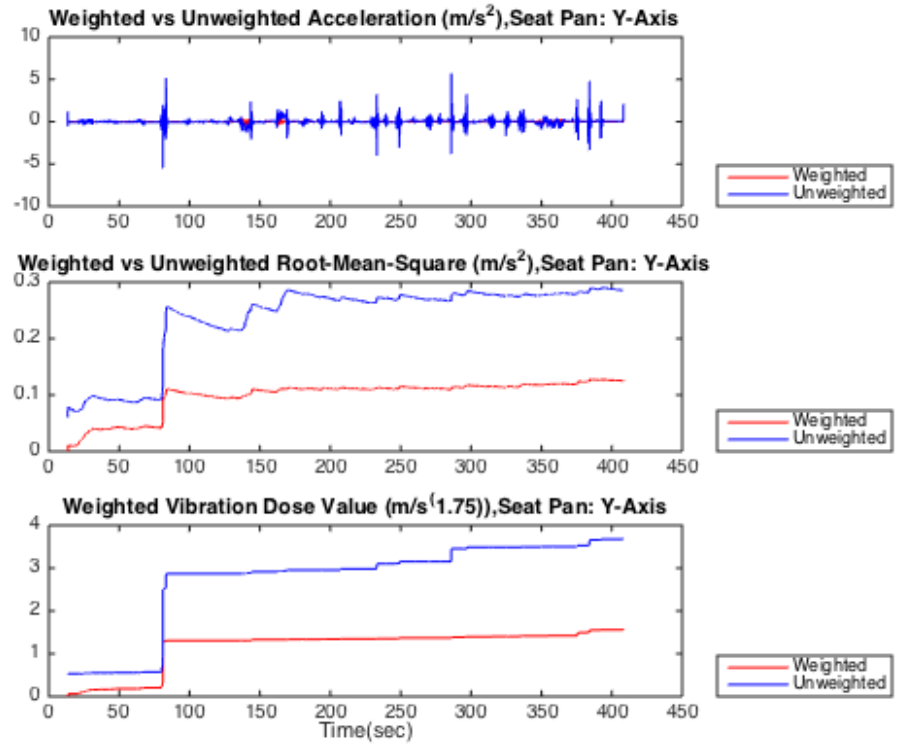

Unweighted Acceleration (m/s2),Seat Pan: X-Axis
Unweighted Root-Mean-Square (m/s2),Seat Pan: X-Axis
Unweighted Vibration Dose Value (m/s(1.75)),Seat Pan: X-Axis
Time(sec)

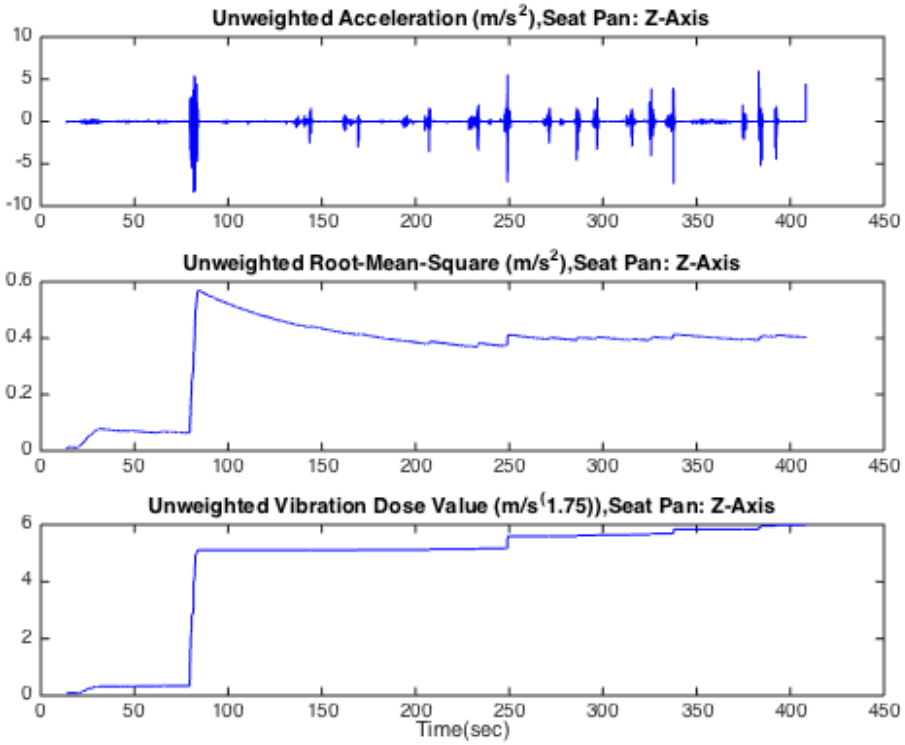

Weighted Acceleration (m/s2),Seat Pan: X-Axis
Weighted Root-Mean-Square (m/s2),Seat Pan: X-Axis
Weighted Vibration Dose Value (m/s(1.75)),Seat Pan: X-Axis
Time(sec)

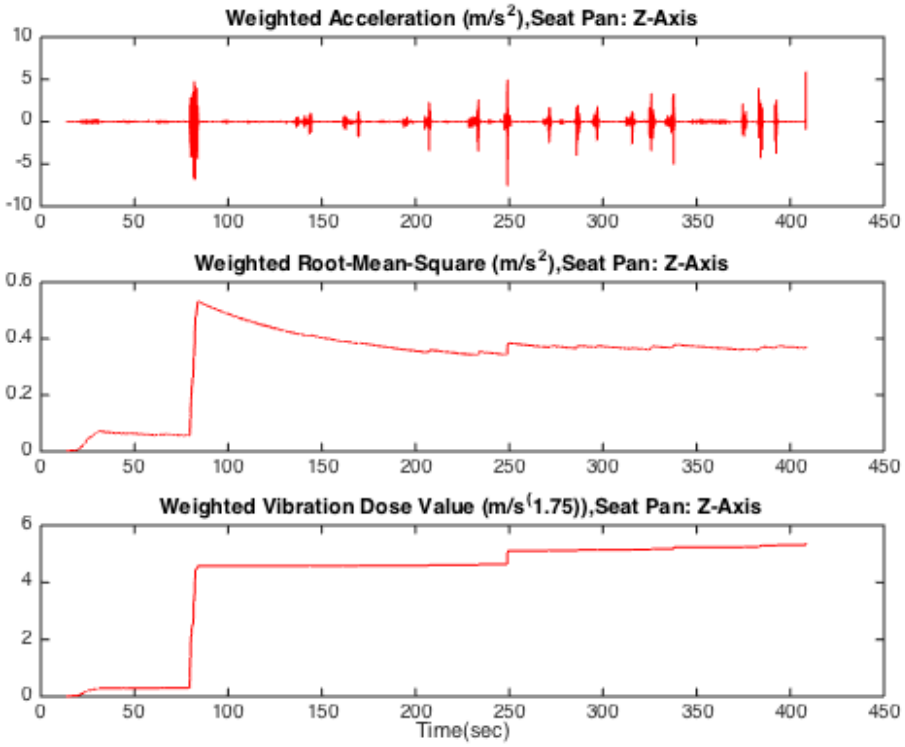

Weighted vs Unweighted Acceleration (m/s2),Seat Pan: X-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s2),Seat Pan: X-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s1.75)),Seat Pan: X-Axis
Weighted
Unweighted
Time(sec)

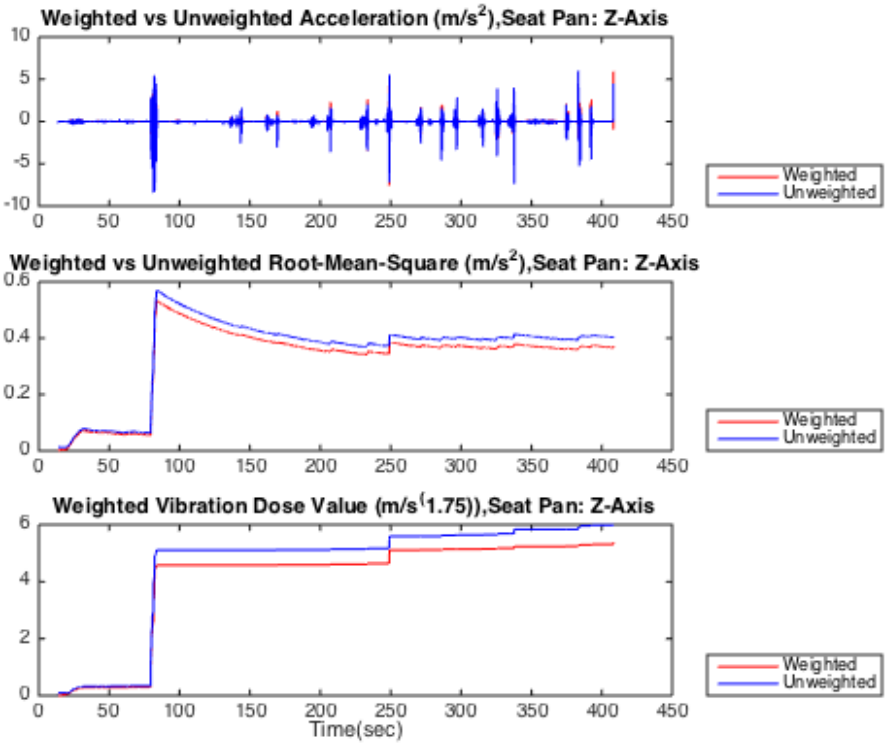

Unweighted Acceleration (m/s2),Seat Pan: Y-Axis
Unweighted Root-Mean-Square (m/s2),Seat Pan: Y-Axis
Unweighted Vibration Dose Value (m/s1.75)),Seat Pan: Y-Axis
Time(sec)


Weighted Acceleration (m/s2),Seat Pan: Y-Axis
Weighted Root-Mean-Square (m/s2),Seat Pan: Y-Axis
Weighted Vibration Dose Value (m/s(1.75)),Seat Pan: Y-Axis
Time(sec)


Weighted vs Unweighted Acceleration (m/s2),Seat Pan: Y-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s2),Seat Pan: Y-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s(1.75)),Seat Pan: Y-Axis
Weighted
Unweighted
Time(sec)


Unweighted Acceleration (m/s2),Seat Pan: Z-Axis
Unweighted Root-Mean-Square (m/s2),Seat Pan: Z-Axis
Unweighted Vibration Dose Value (m/s(1.75)),Seat Pan: Z-Axis
Time(sec)


Weighted Acceleration (m/s2),Seat Pan: Z-Axis
Weighted Root-Mean-Square (m/s2),Seat Pan: Z-Axis
Weighted Vibration Dose Value (m/s(1.75)),Seat Pan: Z-Axis
Time(sec)




```
if a==1 && m==1
    tim_bx = tim;
    amp_bx=amp;
    aurms_run_bx=aurms_run;
    VDVu_run_bx=VDVu_run;
    aw_bx=aw;
    awrms_run_bx=awrms_run;
    VDVw_run_bx=VDVw_run;
    VDVw_bx=VDVw;
elseif a==1 && m==2
    tim_by = tim;
    amp_by=amp;
    aurms_run_by=aurms_run;
    VDVu_run_by=VDVu_run;
    aw_by=aw;
    awrms_run_by=awrms_run;
    VDVw_run_by=VDVw_run;
    VDVw_by=VDVw;
elseif a==1 && m==3
    tim_bz = tim;
```

```
            amp_bz=amp;
            aurms_run_bz=aurms_run;
            VDVu_run_bz=VDVu_run;
            aw_bz=aw;
            awrms_run_bz=awrms_run;
            VDVw_run_bz=VDVw_run;
            VDVw_bz=VDVw;
        elseif a==2 && m==1
            tim_fx = tim;
            amp_bx=amp;
            aurms_run_fx=aurms_run;
            VDVu_run_fx=VDVu_run;
            aw_fx=aw;
            awrms_run_fx=awrms_run;
            VDVw_run_fx=VDVw_run;
            VDVw_fx=VDVw;
        elseif a==2 && m==2
            tim_fy = tim;
            amp_fy=amp;
            aurms_run_fy=aurms_run;
            VDVu_run_fy=VDVu_run;
            aw_fy=aw;
            awrms_run_fy=awrms_run;
            VDVw_run_fy=VDVw_run;
            VDVw_fy=VDVw;
        elseif a==2 && m==3
            tim_fz = tim;
            amp_fz=amp;
            aurms_run_fz=aurms_run;
            VDVu_run_fz=VDVu_run;
            aw_fz=aw;
            awrms_run_fz=awrms_run;
            VDVw_run_fz=VDVw_run;
            VDVw_fz=VDVw;
        elseif a==3 && m==1
            tim_px = tim;
            amp_px=amp;
            aurms_run_px=aurms_run;
```

```
            VDVu_run_px=VDVu_run;
            aw_px=aw;
            awrms_run_px=awrms_run;
            VDVw_run_px=VDVw_run;
            VDVw_px=VDVw;
        elseif a==3 && m==2
            tim_py = tim;
            amp_py=amp;
            aurms_run_py=aurms_run;
            VDVu_run_py=VDVu_run;
            aw_py=aw;
            awrms_run_py=awrms_run;
            VDVw_run_py=VDVw_run;
            VDVw_py=VDVw;
        elseif a==3 && m==3
            tim_pz = tim;
            amp_pz=amp;
            aurms_run_pz=aurms_run;
            VDVu_run_pz=VDVu_run;
            aw_pz=aw;
            awrms_run_pz=awrms_run;
            VDVw_run_pz=VDVw_run;
            VDVw_pz=VDVw;
        end
 end
    %
    if a==1
        VDVw_bxyz=((VDVw_bx^4)+(VDVw_by^4)+(VDVw_bz^4))^(.25)
        VDVw_bxyz_Health=((kxh^4)*(VDVw_bx^4)+(kyh^4)*(VDVw_by^4)+
(kzh^4)*(VDVw_bz^4))^(.25)
        VDVw_bxyz_Comfort=((kxc^4)*(VDVw_bx^4)+(kyc^4)*(VDVw_by^4)+
(kzc^4)*(VDVw_bz^4))^(.25)
        RMSw_bxyz=((kxh^2)*(awrms_bx^2)+(kyh^2)*(awrms_by^2)+(kzh^2)*(awrms_bz^2))^(.
5)
    elseif a==2
        VDVw_fxyz=((VDVw_fx^4)+(VDVw_fy^4)+(VDVw_fz^4))^(.25)
        VDVw_fxyz_Health=((kxh^4)*(VDVw_fx^4)+(kyh^4)*(VDVw_fy^4)+
(kzh^4)*(VDVw_fz^4))^(.25)
```

```
        VDVw_fxyz_Comfort=((kxc^4)*(VDVw_fx^4)+(kyc^4)*(VDVw_fy^4)+
(kzc^4)*(VDVw_fz^4))^(.25)
        RMSw_fxyz=((kxh^2)*(awrms_fx^2)+(kyh^2)*(awrms_fy^2)+(kzh^2)*(awrms_fz^2))^(.
5)
    elseif a==3
        VDVw_pxyz=((VDVw_px^4)+(VDVw_py^4)+(VDVw_pz^4))^(.25)
        VDVw_pxyz_Health=((kxh^4)*(VDVw_px^4)+(kyh^4)*(VDVw_py^4)+
(kzh^4)*(VDVw_pz^4))^(.25)
        VDVw_pxyz_Comfort=((kxc^4)*(VDVw_px^4)+(kyc^4)*(VDVw_py^4)+
(kzc^4)*(VDVw_pz^4))^(.25)
        RMSw_pxyz=((kxh^2)*(awrms_px^2)+(kyh^2)*(awrms_py^2)+(kzh^2)*(awrms_pz^2))^(.
5)
    end
end



  Based on th_weighted.m  ver 1.2  May 31, 2013 by Tom Irvine  Email:
tom@vibrationdata.com
  Adapthed by LNard Tufts  March 24, 2015


  This program converts an acceleration time history to a
  weighted format per ISO 2631.



VDVw_bxyz =

   20.3579


VDVw_bxyz_Health =

   16.2745


VDVw_bxyz_Comfort =
```

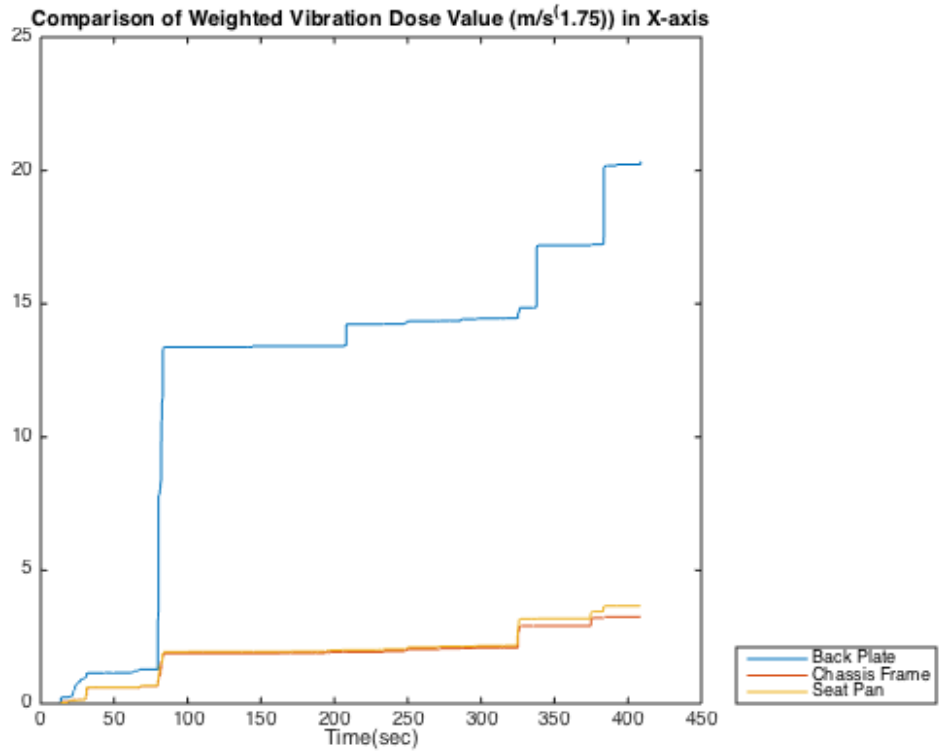
```
   16.2745

RMSw_bxyz =

    0.9954

VDVw_fxyz =

    7.3552

VDVw_fxyz_Health =

    7.5764

VDVw_fxyz_Comfort =

    7.3552

RMSw_fxyz =

    0.6718

VDVw_pxyz =

    5.6651

VDVw_pxyz_Health =

    6.2980
```

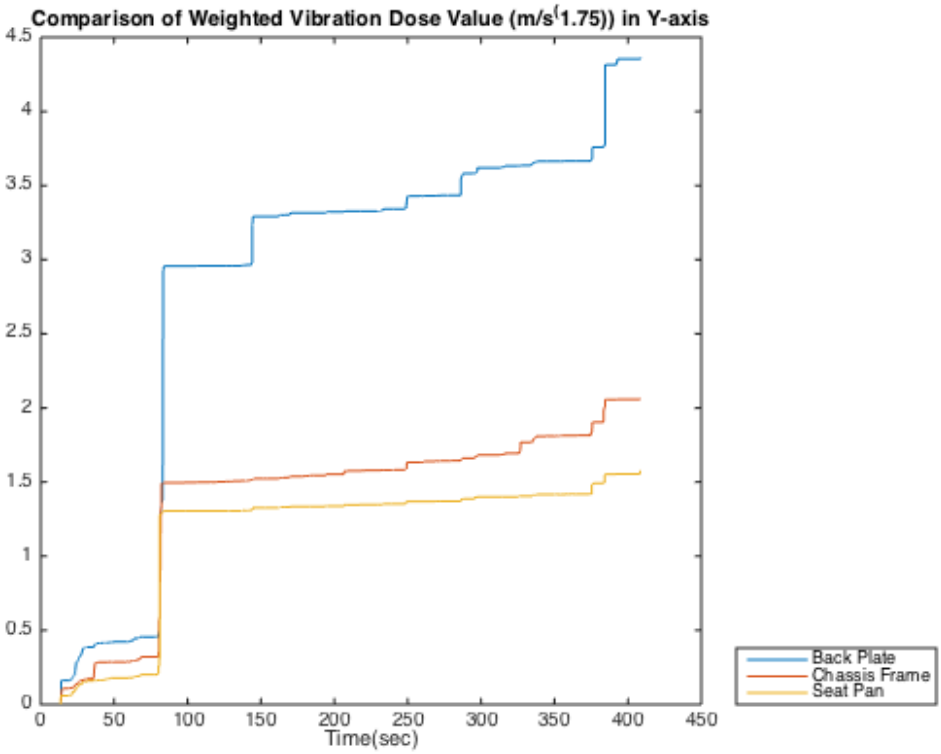
```
VDVw_pxyz_Comfort =

    5.6651


RMSw_pxyz =

    0.5967
```

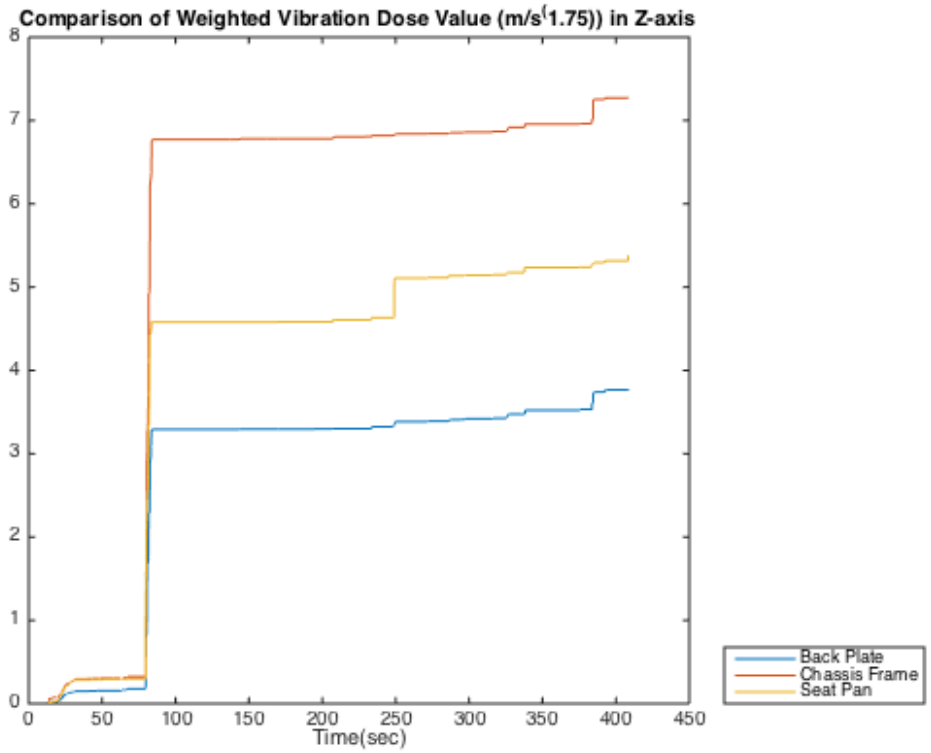
```
figure
plot(tim_bx,VDVw_run_bx,tim_fx,VDVw_run_fx,tim_px,VDVw_run_px)
title('Comparison of Weighted Vibration Dose Value (m/s^(1.75)) in X-axis')
legend('Back Plate','Chassis Frame','Seat Pan','Location','SouthEastOutside')
xlabel('Time(sec)');

out1=sprintf('\n\n Chassis Frame to Seat Pan SEAT Value in X-Axis  SEAT =
%3.1f',VDVw_px*100/VDVw_fx);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support SEAT Value in X-Axis  SEAT =
%3.1f',VDVw_bx*100/VDVw_fx);
disp(out1);
disp('**********************************************************************
***************')
%
figure
plot(tim_by,VDVw_run_by,tim_fy,VDVw_run_fy,tim_py,VDVw_run_py)
title('Comparison of Weighted Vibration Dose Value (m/s^(1.75)) in Y-axis')
legend('Back Plate','Chassis Frame','Seat Pan','Location','SouthEastOutside')
xlabel('Time(sec)');

out1=sprintf('\n\n Chassis Frame to Seat Pan SEAT Value in Y-Axis  SEAT =
%3.1f',VDVw_py*100/VDVw_fy);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support SEAT Value in Y-Axis  SEAT =
%3.1f',VDVw_by*100/VDVw_fy);
disp(out1);
```

```
disp('*********************************************************************************
****************')
%
figure
plot(tim_bz,VDVw_run_bz,tim_fz,VDVw_run_fz,tim_pz,VDVw_run_pz)
title('Comparison of Weighted Vibration Dose Value (m/s^(1.75)) in Z-axis')
legend('Back Plate','Chassis Frame','Seat Pan','Location','SouthEastOutside')
xlabel('Time(sec)');

out1=sprintf('\n\n Chassis Frame to Seat Pan SEAT Value in Z-Axis  SEAT =
%3.1f',VDVw_pz*100/VDVw_fz);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support SEAT Value in Z-Axis  SEAT =
%3.1f',VDVw_bz*100/VDVw_fz);
disp(out1);
disp('*********************************************************************************
****************')
```

```
 Chassis Frame to Seat Pan SEAT Value in X-Axis  SEAT = 113.5


 Chassis Frame to Back Support SEAT Value in X-Axis  SEAT = 625.8
*********************************************************************************


 Chassis Frame to Seat Pan SEAT Value in Y-Axis  SEAT = 76.7


 Chassis Frame to Back Support SEAT Value in Y-Axis  SEAT = 212.0
*********************************************************************************


 Chassis Frame to Seat Pan SEAT Value in Z-Axis  SEAT = 74.0


 Chassis Frame to Back Support SEAT Value in Z-Axis  SEAT = 51.9
*********************************************************************************
```


Comparison of Weighted Vibration Dose Value (m/s^1.75)) in X-axis
25
20
15
10
5
0
0
50
100
150
200
250
300
350
400
450
Time(sec)
Back Plate
Chassis Frame
Seat Pan


Comparison of Weighted Vibration Dose Value (m/s^1.75)) in Y-axis
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
50
100
150
200
250
300
350
400
450
Time(sec)
Back Plate
Chassis Frame
Seat Pan




```
out1=sprintf('\n\n Chassis Frame to Seat Pan Overall SEAT Value  SEAT =
%3.1f',VDVw_pxyz*100/VDVw_fxyz);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support Overall SEAT Value  SEAT =
%3.1f',VDVw_bxyz*100/VDVw_fxyz);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Seat Pan Overall SEAT Value (Health Weighted)
SEAT = %3.1f',VDVw_pxyz_Health*100/VDVw_fxyz_Health);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support Overall SEAT Value (Health Weighted)
SEAT = %3.1f',VDVw_bxyz_Health*100/VDVw_fxyz_Health);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Seat Pan Overall SEAT Value (Comfort Weighted)
SEAT = %3.1f',VDVw_pxyz_Comfort*100/VDVw_fxyz_Comfort);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support Overall SEAT Value (Comfort Weighted)
SEAT = %3.1f',VDVw_bxyz_Comfort*100/VDVw_fxyz_Comfort);
disp(out1);

 Chassis Frame to Seat Pan Overall SEAT Value  SEAT = 77.0
```

```
Chassis Frame to Back Support Overall SEAT Value  SEAT = 276.8

Chassis Frame to Seat Pan Overall SEAT Value (Health Weighted)  SEAT = 83.1

Chassis Frame to Back Support Overall SEAT Value (Health Weighted)  SEAT = 214.8

Chassis Frame to Seat Pan Overall SEAT Value (Comfort Weighted)  SEAT = 77.0

Chassis Frame to Back Support Overall SEAT Value (Comfort Weighted)  SEAT = 221.3
```

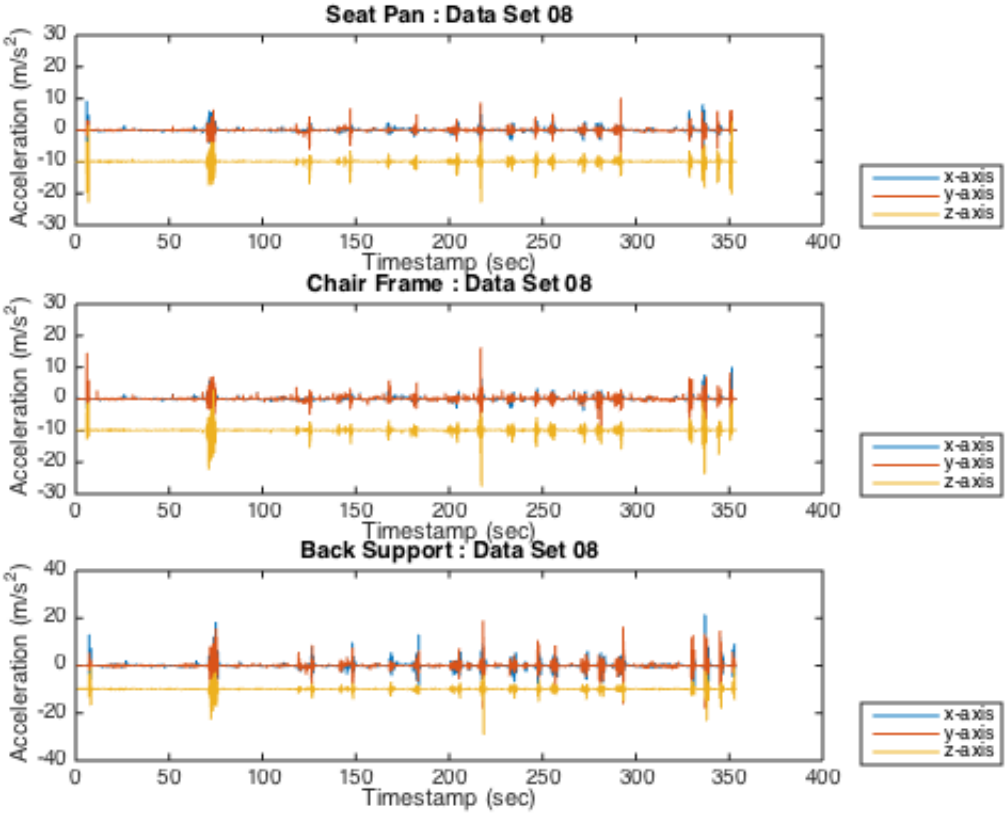
Published with MATLAB® R2014b

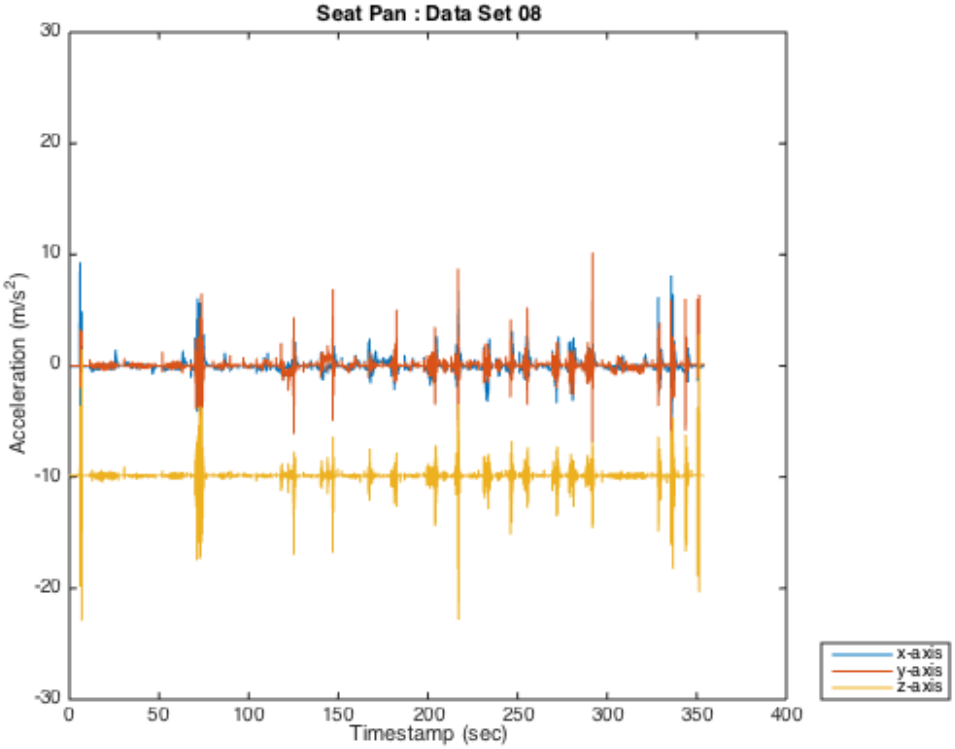
APPENDIX O

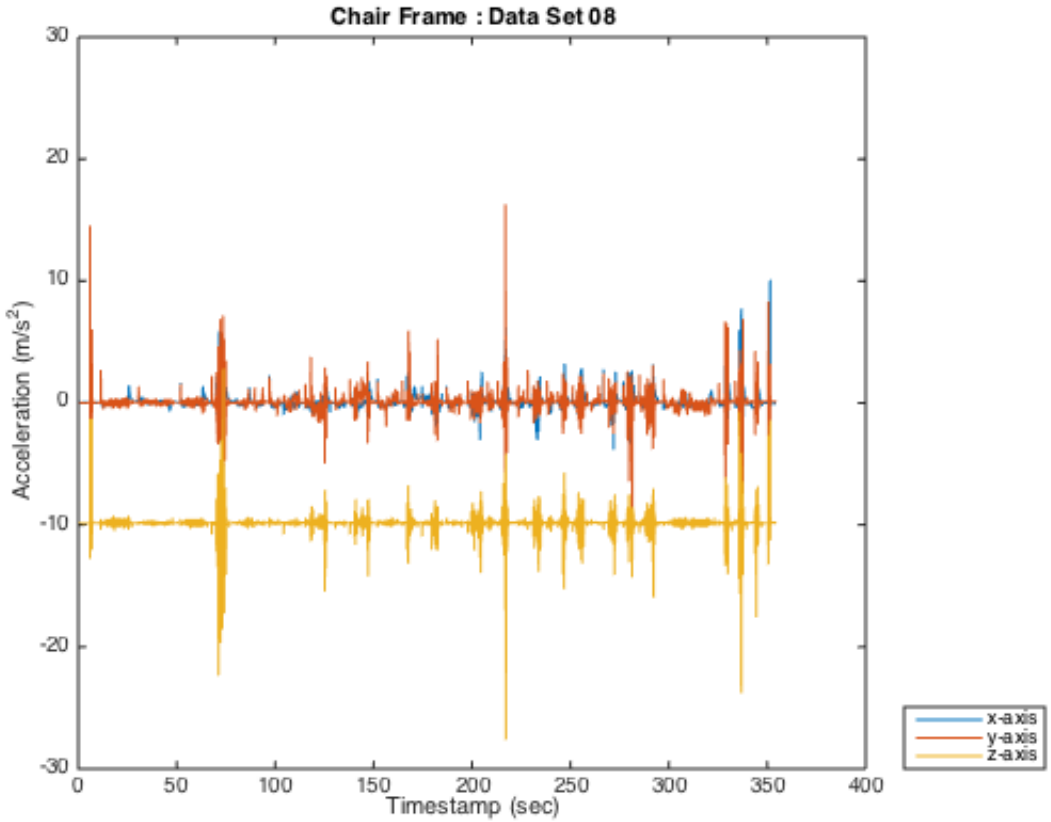
MATLab Data Processing and Analysis Script Data Set 08

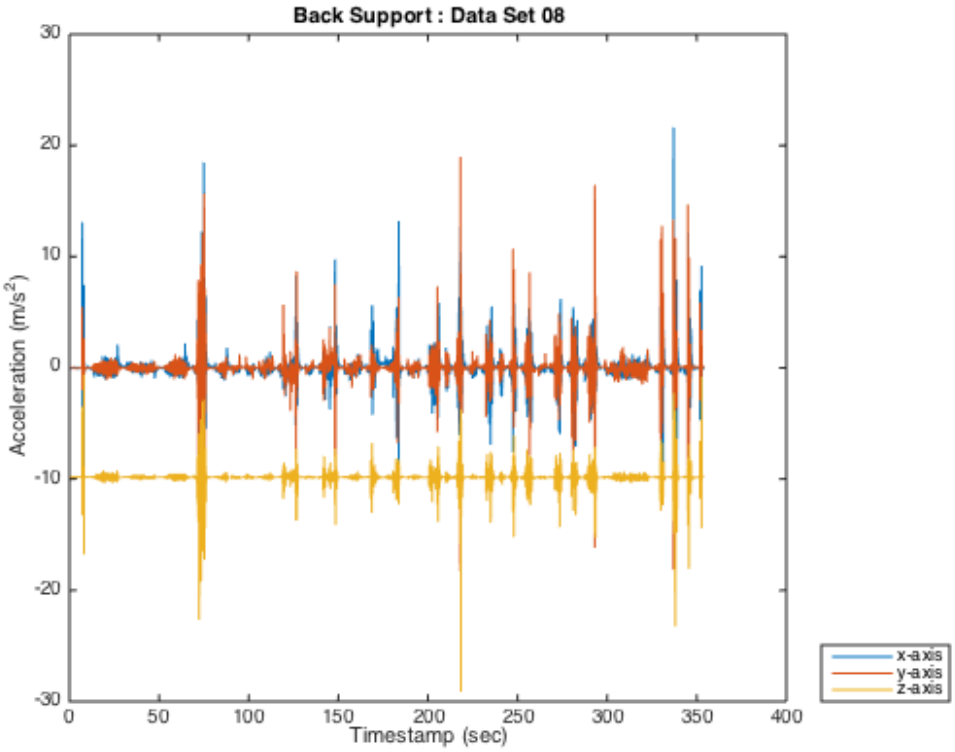
```
close all;
load('DataSet08.mat')

x_b_off = -.54;
y_b_off = +.29;
z_b_off = -.43;

x_p_off = +.19;
y_p_off = +.78;
z_p_off = +.77;


x_f_off = +.28;
y_f_off = +.49;
z_f_off = +.09;


x_b = -((ds8_back_z) + (z_b_off));
y_b = ds8_back_x + x_b_off;
z_b = -((ds8_back_y) + (y_b_off));
t_b = ds8_back_t/1000;

x_p = ds8_pan_y + y_p_off;
y_p = ds8_pan_x + (x_p_off);
z_p = -((ds8_pan_z)+ (z_p_off));
t_p = ds8_pan_t/1000;

x_f = -((ds8_frame_y) + (y_f_off));
y_f = ds8_frame_x + x_f_off;
z_f = ds8_frame_z + z_f_off;
t_f = ds8_frame_t/1000;
figure(1)
subplot (3,1,1)
plot(t_p,x_p,t_p,y_p,t_p,z_p)
ylim([-30 30])
title('Seat Pan : Data Set 08')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')

subplot (3,1,2)
plot(t_f,x_f,t_f,y_f,t_f,z_f)
ylim([-30 30])
title('Chair Frame : Data Set 08')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')

subplot (3,1,3)
ylim([-30 30])
plot(t_b,x_b,t_b,y_b,t_b,z_b)
title('Back Support : Data Set 08')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')
```

```
figure(2)
plot(t_p,x_p,t_p,y_p,t_p,z_p)
ylim([-30 30])
title('Seat Pan : Data Set 08')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')

figure(3)
plot(t_f,x_f,t_f,y_f,t_f,z_f)
ylim([-30 30])
title('Chair Frame : Data Set 08')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')

figure(4)
plot(t_b,x_b,t_b,y_b,t_b,z_b)
ylim([-30 30])
title('Back Support : Data Set 08')
xlabel('Timestamp (sec)')
ylabel('Acceleration (m/s^2)')
legend('x-axis','y-axis','z-axis','Location','SouthEastOutside')
hold off
```


Seat Pan : Data Set 08
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis
Chair Frame : Data Set 08
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis
Back Support : Data Set 08
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis


Seat Pan : Data Set 08
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis


Chair Frame : Data Set 08
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis


Back Support : Data Set 08
Acceleration (m/s^2)
Timestamp (sec)
x-axis
y-axis
z-axis

```
%m = 1
back_x_mat = zeros(length(t_b),2);
back_x_mat(:,1) = t_b';
back_x_mat(:,2) = x_b';

%m = 2
back_y_mat = zeros(length(t_b),2);
back_y_mat(:,1) = t_b';
back_y_mat(:,2) = y_b';

%m = 3
back_z_mat = zeros(length(t_b),2);
back_z_mat(:,1) = t_b';
back_z_mat(:,2) = z_b';

%m = 4
frame_x_mat = zeros(length(t_f),2);
frame_x_mat(:,1) = t_f';
frame_x_mat(:,2) = x_f';

%m = 5
frame_y_mat = zeros(length(t_f),2);
frame_y_mat(:,1) = t_f';
frame_y_mat(:,2) = y_f';

%m = 6
frame_z_mat = zeros(length(t_f),2);
frame_z_mat(:,1) = t_f';
frame_z_mat(:,2) = z_f';

%m = 7
pan_x_mat = zeros(length(t_p),2);
pan_x_mat(:,1) = t_p';
pan_x_mat(:,2) = x_p';

%m = 8
pan_y_mat = zeros(length(t_p),2);
pan_y_mat(:,1) = t_p';
pan_y_mat(:,2) = y_p';

%m = 9
pan_z_mat = zeros(length(t_p),2);
pan_z_mat(:,1) = t_p';
pan_z_mat(:,2) = z_p';
disp(' ');
disp('    ');
disp(' ');
disp('  Based on th_weighted.m  ver 1.2  May 31, 2013 by Tom Irvine  Email:
tom@vibrationdata.com ');
disp('  Adapthed by LNard Tufts  March 24, 2015 ');
disp('  ');
disp('  ');
disp('  This program converts an acceleration time history to a ');
disp('  weighted format per ISO 2631. ');
disp('  ');
disp(' ');
%
for a=1:3
```

```
    for m=1:3
        clear amp;
        clear f;
        clear length;
        clear THM;
        clear t;
        clear f;
        clear y;
        clear ww;
        clear matrix_name;

        fig_num=5;
        %[t,f,dt,sr,tmx,tmi,~,ncontinue]=enter_time_history();
        %
        %disp(' Select file input method ');
        %disp('   1=external ASCII file ');
        %disp('   2=file preloaded into Matlab ');
        %disp('   3=Excel file ');
        %file_choice = input('');
        file_choice=2;
        %
        if(file_choice==1)
            [filename, pathname] = uigetfile('*.*');
            filename = fullfile(pathname, filename);
            fid = fopen(filename,'r');
            THM = fscanf(fid,'%g %g',[2 inf]);
            THM=THM';
        end
        if(file_choice==2)
            %FS = input(' Enter the matrix name:  ','s');
            if a==1;
                clear THM;
                name={'Back Plate: X-Axis';'Back Plate: Y-Axis';'Back Plate: Z-Axis'};
                kxh=0.8; kyh=0.5; kzh=0.4;
                kxc=0.8; kyc=0.5; kzc=0.4;

                matrix_b(:,:,1) = back_x_mat;
                start_time(1) = 8.23;
                end_time(1) = 351.8;
                weighting(1) = 4;

                matrix_b(:,:,2) = back_y_mat;
                start_time(2) = 8.23;
                end_time(2) = 351.8;
                weighting(2) = 2;

                matrix_b(:,:,3) = back_z_mat;
                start_time(3) = 8.23;
                end_time(3) = 351.8;
                weighting(3) = 2;
                THM = matrix_b(:,:,m);
            elseif a==2;
                clear THM;
                name={'Chassis Frame: X-Axis';'Chassis Frame: Y-Axis';'Chassis Frame:
Z-Axis'};
                kxh=1.4; kyh=1.4; kzh=1;
                kxc=1; kyc=1; kzc=1;
```

```
                matrix_f(:,:,1) = frame_x_mat;
                start_time(1) = 7.127;
                end_time(1) = 350.4;
                weighting(1) = 2;

                matrix_f(:,:,2) = frame_y_mat;
                start_time(2) = 7.127;
                end_time(2) = 350.4;
                weighting(2) = 2;

                matrix_f(:,:,3) = frame_z_mat;
                start_time(3) = 7.127;
                end_time(3) = 350.4;
                weighting(3) = 1;
                THM = matrix_f(:,:,m);
            elseif a==3;
                clear THM;
                name={'Seat Pan: X-Axis';'Seat Pan: Y-Axis';'Seat Pan: Z-Axis'};
                kxh=1.4; kyh=1.4; kzh=1;
                kxc=1; kyc=1; kzc=1;

                matrix_p(:,:,1) = pan_x_mat;
                start_time(1) = 7.127;
                end_time(1) = 350.4;
                weighting(1) = 2;

                matrix_p(:,:,2) = pan_y_mat;
                start_time(2) = 7.127;
                end_time(2) = 350.4;
                weighting(2) = 2;

                matrix_p(:,:,3) = pan_z_mat;
                start_time(3) = 7.127;
                end_time(3) = 350.4;
                weighting(3) = 1;
                THM = matrix_p(:,:,m);
            end
            %HM=evalin('caller',FS);
            %THM = FS;
        end
        if(file_choice==3)
            [filename, pathname] = uigetfile('*.*');
            xfile = fullfile(pathname, filename);
            %
            THM = xlsread(xfile);
            %
        end
        t=THM(:,1);
        f=THM(:,2);

        %
        tmx=max(t);
        tmi=min(t);
        n = length(f);
        %dt=(tmx-tmi)/(n-1);
        for i=1:(n-1)
            dt_run(i,1)=t(i+1)-t(i);
        end
```

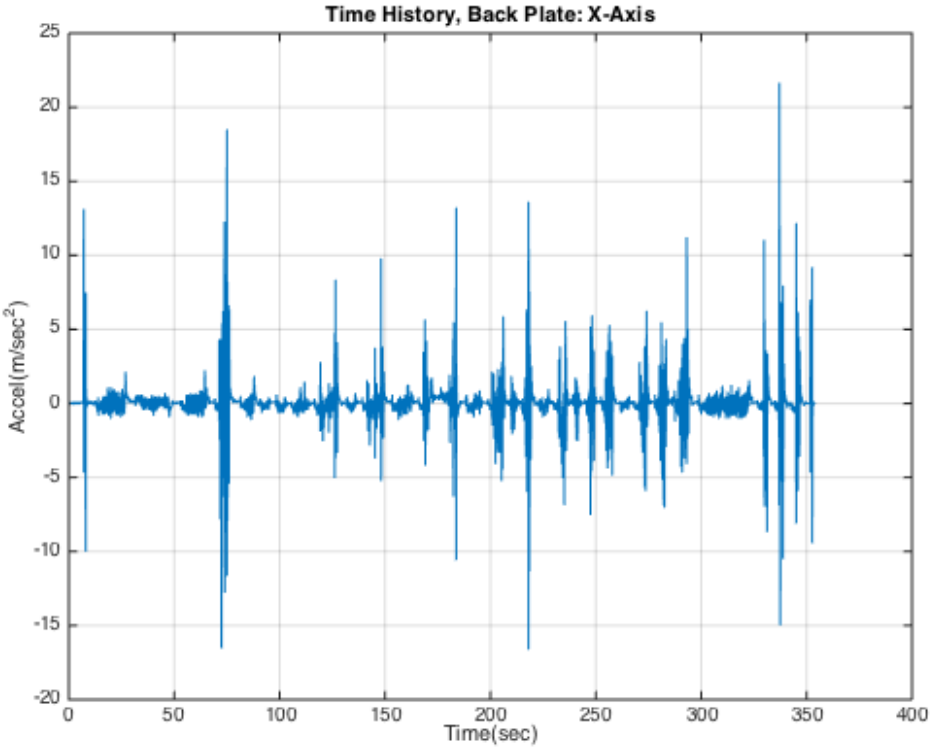
```
        dt=mean(dt_run);
        sr=1./dt;
        %

        disp('************************************************************')
        disp(' ')

        disp(name{m})

        disp(' ')
        disp(' Time Step ');
        %dtmin=min(diff(t));
        %dtmax=max(diff(t));
        dtmin=min(dt_run);
        dtmax=max(dt_run);
        %
        out4 = sprintf(' dtmin  = %8.4g sec  ',dtmin);
        out5 = sprintf(' dt     = %8.4g sec  ',dt);
        out6 = sprintf(' dtmax  = %8.4g sec  ',dtmax);
        disp(out4)
        disp(out5)
        disp(out6)
        %
        disp(' ')
        disp(' Sample Rate ');
        out4 = sprintf(' srmin  = %8.4g samples/sec  ',1/dtmax);
        out5 = sprintf(' sr     = %8.4g samples/sec  ',1/dt);
        out6 = sprintf(' srmax  = %8.4g samples/sec  \n',1/dtmin);
        disp(out4)
        disp(out5)
        disp(out6)
        %
        ncontinue=1;
        if(((dtmax-dtmin)/dt)>0.01)
            %disp(' ')
            %disp(' Warning:  time step is not constant.  Continue calculation? 1=yes
2=no ')
            %ncontinue=input(' ');
            ncontinue=1;
        end
************************************************************

Back Plate: X-Axis

 Time Step
 dtmin  =    0.005 sec
 dt     = 0.006914 sec
 dtmax  =    0.193 sec

 Sample Rate
 srmin  =    5.181 samples/sec
 sr     =    144.6 samples/sec
 srmax  =      200 samples/sec

************************************************************

Back Plate: Y-Axis
```

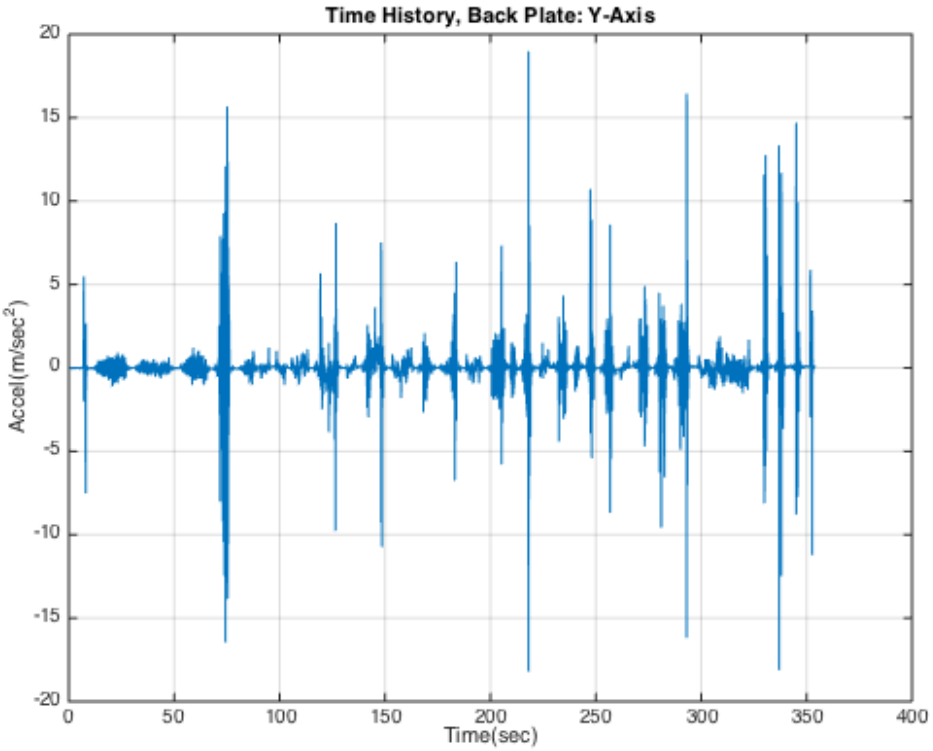
```
 Time Step
 dtmin  =    0.005 sec
 dt     = 0.006914 sec
 dtmax  =    0.193 sec

 Sample Rate
 srmin  =    5.181 samples/sec
 sr     =    144.6 samples/sec
 srmax  =      200 samples/sec

****************************************************************

Back Plate: Z-Axis

 Time Step
 dtmin  =    0.005 sec
 dt     = 0.006914 sec
 dtmax  =    0.193 sec

 Sample Rate
 srmin  =    5.181 samples/sec
 sr     =    144.6 samples/sec
 srmax  =      200 samples/sec

****************************************************************

Chassis Frame: X-Axis

 Time Step
 dtmin  =    0.005 sec
 dt     = 0.006864 sec
 dtmax  =    0.427 sec

 Sample Rate
 srmin  =    2.342 samples/sec
 sr     =    145.7 samples/sec
 srmax  =      200 samples/sec

****************************************************************

Chassis Frame: Y-Axis

 Time Step
 dtmin  =    0.005 sec
 dt     = 0.006864 sec
 dtmax  =    0.427 sec

 Sample Rate
 srmin  =    2.342 samples/sec
 sr     =    145.7 samples/sec
 srmax  =      200 samples/sec

****************************************************************

Chassis Frame: Z-Axis

 Time Step
 dtmin  =    0.005 sec
```

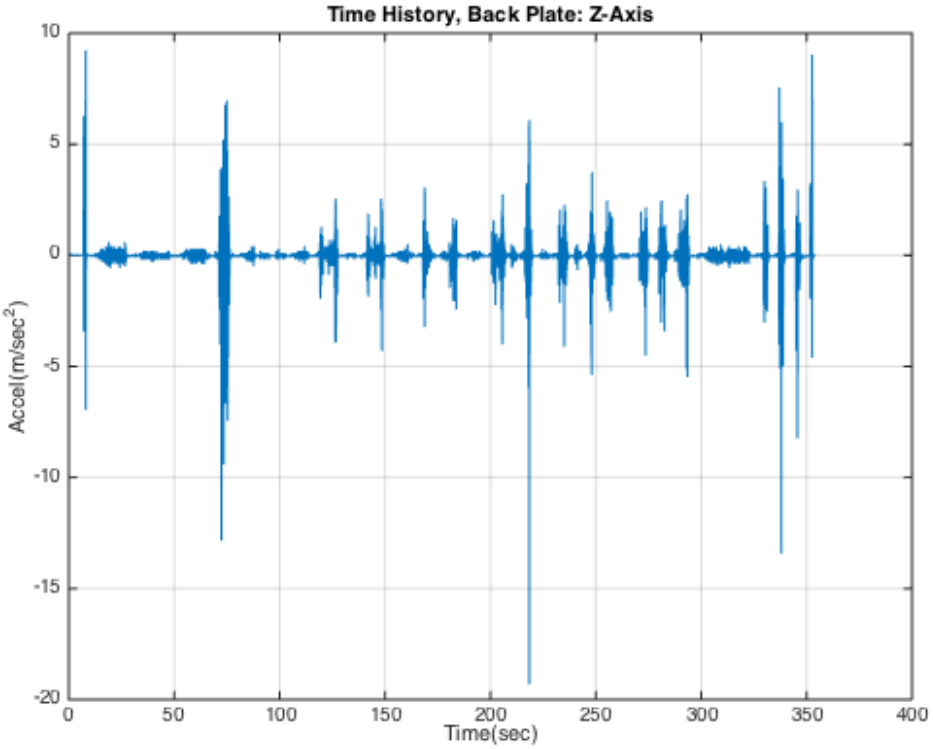
```
 dt     = 0.006864 sec
 dtmax  =    0.427 sec

 Sample Rate
 srmin  =    2.342 samples/sec
 sr     =    145.7 samples/sec
 srmax  =      200 samples/sec

*****************************************************************

Seat Pan: X-Axis

 Time Step
 dtmin  =    0.005 sec
 dt     = 0.006746 sec
 dtmax  =    0.326 sec

 Sample Rate
 srmin  =    3.067 samples/sec
 sr     =    148.2 samples/sec
 srmax  =      200 samples/sec

*****************************************************************

Seat Pan: Y-Axis

 Time Step
 dtmin  =    0.005 sec
 dt     = 0.006746 sec
 dtmax  =    0.326 sec

 Sample Rate
 srmin  =    3.067 samples/sec
 sr     =    148.2 samples/sec
 srmax  =      200 samples/sec

*****************************************************************

Seat Pan: Z-Axis

 Time Step
 dtmin  =    0.005 sec
 dt     = 0.006746 sec
 dtmax  =    0.326 sec

 Sample Rate
 srmin  =    3.067 samples/sec
 sr     =    148.2 samples/sec
 srmax  =      200 samples/sec
```

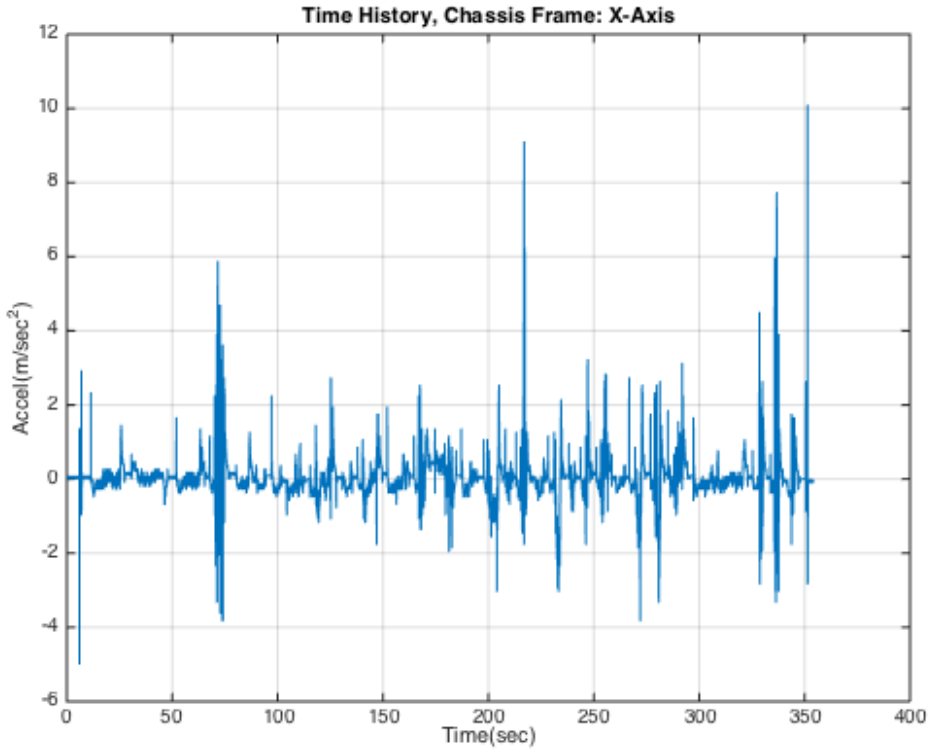
```
        %disp(' ');
        %disp(' Remove mean?  1=yes 2=no');
        %
        %imr=input(' ');
        imr=1;
        if(imr==1)
            f=f-mean(f);
        end
```

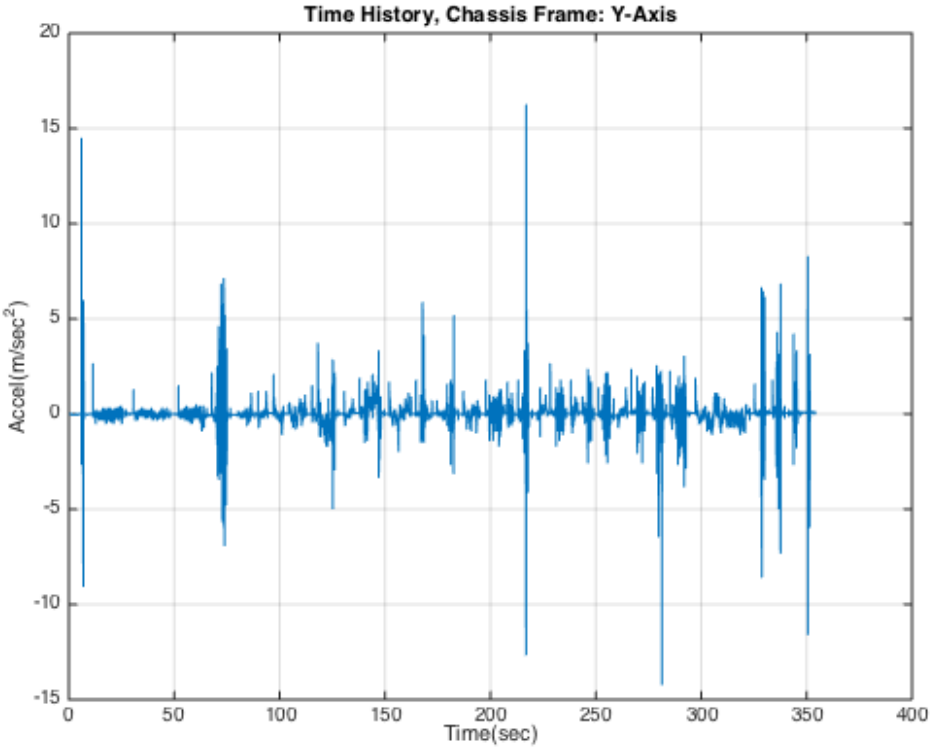
```
%
THM=[t f];
%
iunits=2;
%
while( iunits ~= 1 && iunits ~=2 )
    %
    out1=sprintf('\n Enter input unit:\n  1=G  2= m/sec^2  ');
    disp(out1);
    iunits=input(' ');
    %
end
%
if(iunits==1)
    p_unit=sprintf('G');
else
    p_unit=sprintf('m/sec^2');
end
%
x_label=sprintf('Time(sec)');
y_label=sprintf('Accel(%s)',p_unit);
t_string=sprintf(['Time History, ',name{m}]);
[fig_num]=plot_TH(fig_num,x_label,y_label,t_string,THM);
%
%disp(' ');
%st=input(' Enter starting time (sec) ');
st = start_time(m);
%
%disp(' ');
%te=input(' Enter ending time (sec) ');
te = end_time(m);
%
j=1;
jfirst=1;
jlast=max(size(THM));
for i=1:max(size(THM))
    if(THM(i,1)<st)
        jfirst=i;
    end
    if(THM(i,1)>te)
        jlast=i;
        break;
    end
end
%
tim=double(THM(jfirst:jlast,1));
amp=double(THM(jfirst:jlast,2));
%
if(iunits==1)
    amp=amp*9.81;
end
%
if(imr==1)
    amp=amp-mean(amp);
end
```

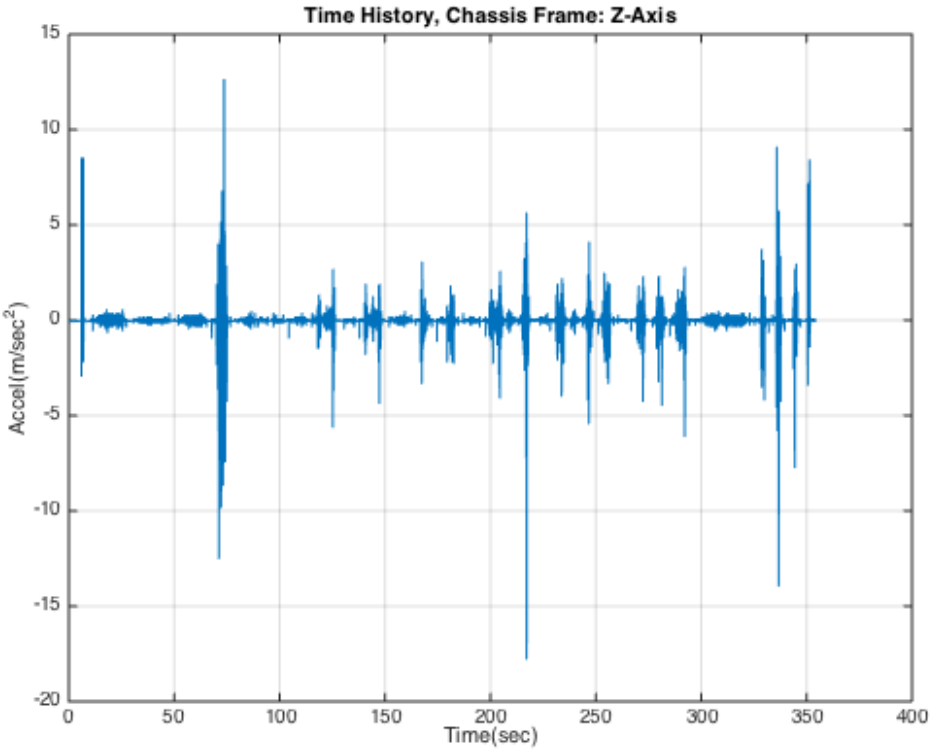

Time History, Back Plate: X-Axis
Accel(m/sec^2)
Time(sec)
25
20
15
10
5
0
-5
-10
-15
-20
0
50
100
150
200
250
300
350
400

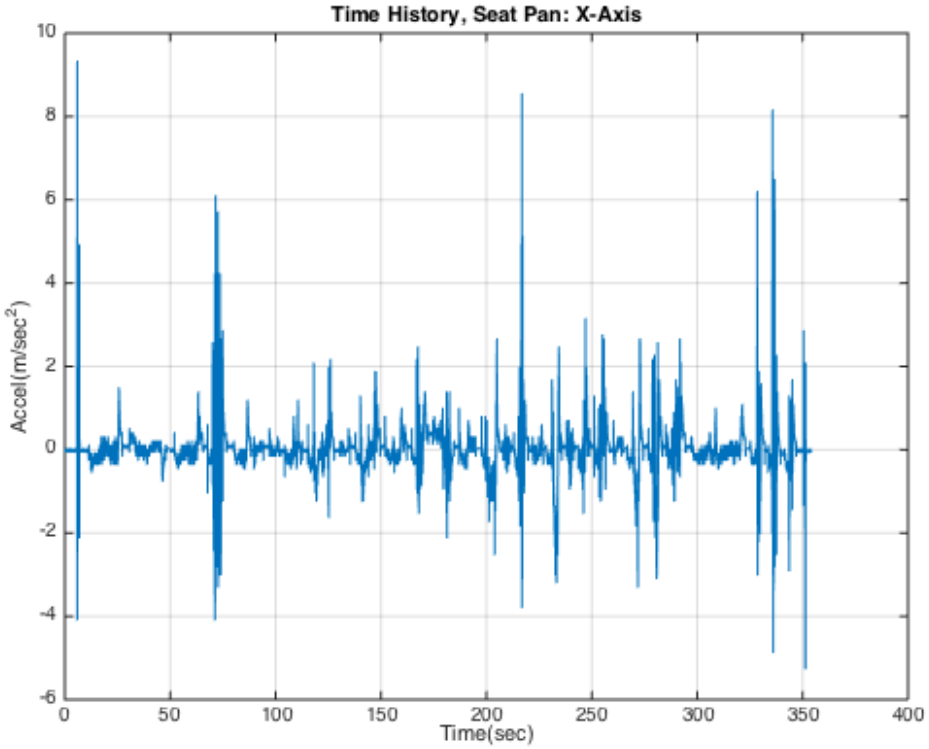

Time History, Back Plate: Y-Axis
Accel(m/sec^2)
Time(sec)
20
15
10
5
0
-5
-10
-15
-20
0
50
100
150
200
250
300
350
400

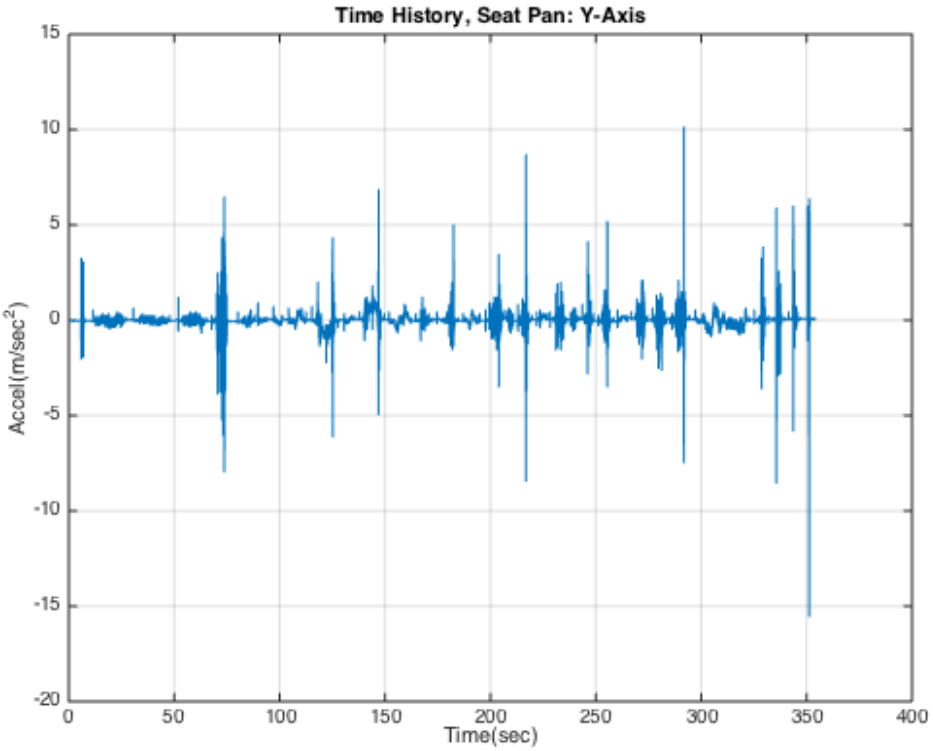

Time History, Back Plate: Z-Axis
Accel(m/sec^2)
Time(sec)
10
5
0
-5
-10
-15
-20
0
50
100
150
200
250
300
350
400

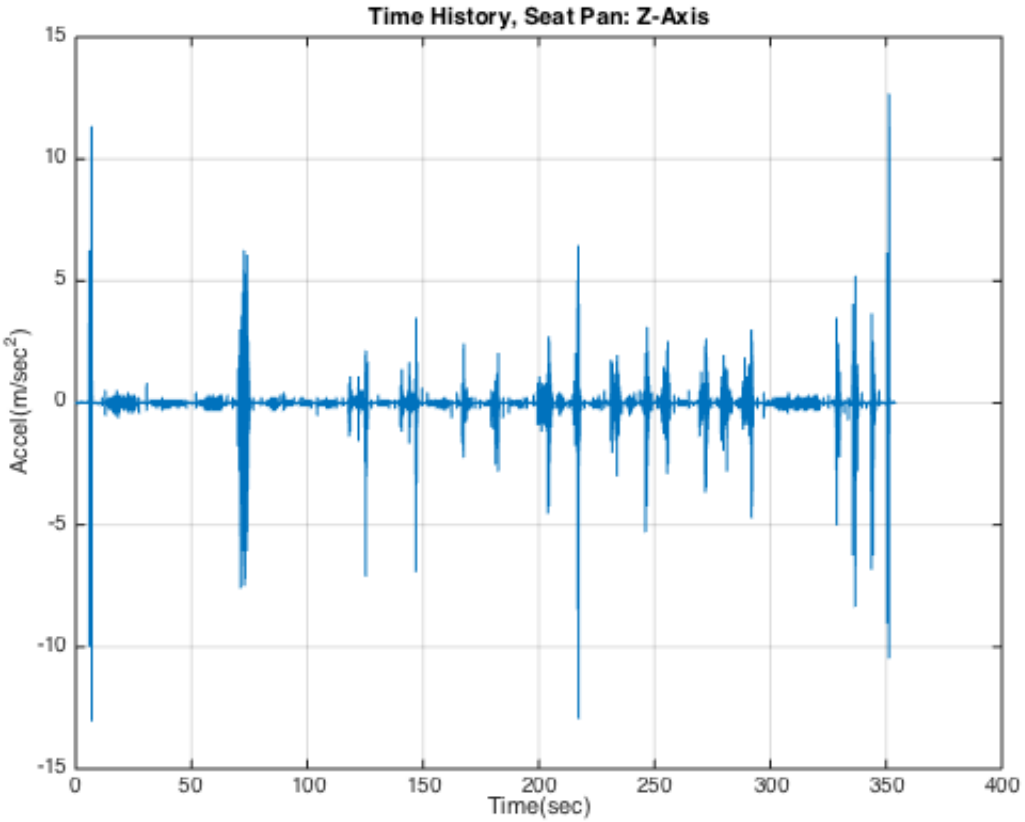

Time History, Chassis Frame: X-Axis
Accel(m/sec^2)
Time(sec)
12
10
8
6
4
2
0
-2
-4
-6
0
50
100
150
200
250
300
350
400


Time History, Chassis Frame: Y-Axis
Accel(m/sec^2)
Time(sec)
20
15
10
5
0
-5
-10
-15
0
50
100
150
200
250
300
350
400


Time History, Chassis Frame: Z-Axis
Accel(m/sec^2)
Time(sec)
15
10
5
0
-5
-10
-15
-20
0
50
100
150
200
250
300
350
400


Time History, Seat Pan: X-Axis
Accel(m/sec^2)
Time(sec)
10
8
6
4
2
0
-2
-4
-6
0
50
100
150
200
250
300
350
400


Time History, Seat Pan: Y-Axis
Accel(m/sec^2)
Time(sec)
15
10
5
0
-5
-10
-15
-20
0
50
100
150
200
250
300
350
400




```
%[fwl,fw,fwu,wk,wd,wf,wc,we,wj,wb,www,iweight]=weight_factors();
%
%
iweight = 0;
%
while( iweight ~= 1 && iweight ~= 2 && iweight ~= 3 && iweight ~= 4 && ...
       iweight ~= 5 && iweight ~= 6 && iweight ~= 7  )
    %
    %disp(' ');
    %disp(' Enter weighting by choice number: ');
    %
    %disp('  1= Wk  z-axis,      seat surface');
    %disp('  2= Wd  x or y-axis, seat surface');
    %disp('  3= Wf');
    %disp('  4= Wc');
    %disp('  5= We');
    %disp('  6= Wj');
    %disp('  7= Wb');
    %disp(' ');
    iweight=weighting(m);
    %
end
%
fc=zeros(44,1);
wk=zeros(44,1);
wd=zeros(44,1);
wf=zeros(44,1);
wc=zeros(44,1);
```

```
        we=zeros(44,1);
        wj=zeros(44,1);
        wb=zeros(44,1);
        %
        %
        fc(1)=0.02;
        wk(1)=-90.;
        wd(1)=-90.;
        wf(1)=-32.33;
        wc(1)=-90.;
        we(1)=-90.;
        wj(1)=-90.;
        wb(1)=-90.;
        %
        fc(2)=0.025;
        wk(2)=-90.;
        wd(2)=-90.;
        wf(2)=-28.48;
        wc(2)=-90.;
        we(2)=-90.;
        wj(2)=-90.;
        wb(2)=-90.;
        %
        fc(3)=0.0315;
        wk(3)=-90.;
        wd(3)=-90.;
        wf(3)=-24.47;
        wc(3)=-90.;
        we(3)=-90.;
        wj(3)= -90.;
        wb(3)=-90.;
        %
        fc(4)=0.04;
        wk(4)=-90.;
        wd(4)=-90.;
        wf(4)=-20.25;
        wc(4)=-90.;
        we(4)=-90.;
        wj(4)=-90.;
        wb(4)=-90.;
        %
        fc(5)=0.05;
        wk(5)=-90.;
        wd(5)=-90.;
        wf(5)=-16.10;
        wc(5)=-90.;
        we(5)=-90.;
        wj(5)=-90.;
        wb(5)=-90.;
        %
        fc(6)=0.063;
        wk(6)=-90.;
        wd(6)=-90.;
        wf(6)=-11.49;
        wc(6)=-90.;
        we(6)= -90.;
        wj(6)=-90.;
        wb(6)=-90.;
```

```
        %
        fc(7)=0.08;
        wk(7)=-90.;
        wd(7)=-90.;
        wf(7)=-6.73;
        wc(7)= -90.;
        we(7)=-90.;
        wj(7)=-90.;
        wb(7)=-90.;
        %
        fc(8)=0.1;
        wk(8)=-30.11;
        wd(8)=-24.09;
        wf(8)=-3.16;
        wc(8)=-24.11;
        we(8)=-24.08;
        wj(8)=-30.18;
        wb(8)=-32.04;
        %
        fc(9)=0.125;
        wk(9)=-26.26;
        wd(9)=-20.24;
        wf(9)=-0.96;
        wc(9)=-20.25;
        we(9)=-20.22;
        wj(9)=-26.32;
        wb(9)=-28.20;
        %
        fc(10)=0.16;
        wk(10)=-22.05;
        wd(10)=-16.01;
        wf(10)=0.05;
        wc(10)=-16.03;
        we(10)=-15.98;
        wj(10)=-22.11;
        wb(10)=-23.98;
        %
        fc(11)=0.2;
        wk(11)=-18.33;
        wd(11)=-12.28;
        wf(11)=-0.07;
        wc(11)=-12.03;
        we(11)=-12.23;
        wj(11)=-18.38;
        wb(11)=-20.23;
        %
        fc(12)=0.25;
        wk(12)=-14.81;
        wd(12)=-8.75;
        wf(12)=-1.37;
        wc(12)=-8.78;
        we(12)=-8.67;
        wj(12)=-14.86;
        wb(12)=-16.71;
        %
        fc(13)=0.315;
        wk(13)=-11.60;
        wd(13)=-5.52;
```

```
        wf(13)=-4.17;
        wc(13)=-5.56;
        we(13)=-5.41;
        wj(13)=-11.65;
        wb(13)=-13.51;
        %
        fc(14)=0.4;
        wk(14)=-9.07;
        wd(14)=-2.94;
        wf(14)=-8.31;
        wc(14)=-3.01;
        we(14)=-3.01;
        wj(14)=-9.10;
        wb(14)=-10.98;
        %
        fc(15)=0.5;
        wk(15)=-7.57;
        wd(15)=-1.38;
        wf(15)=-13.00;
        wc(15)=-1.48;
        we(15)=-1.29;
        wj(15)=-7.60;
        wb(15)=-9.53;
        %
        fc(16)=0.63;
        wk(16)=-6.77;
        wd(16)=-0.50;
        wf(16)=-18.69;
        wc(16)=-0.64;
        we(16)=-0.55;
        wj(16)=-6.78;
        wb(16)=-8.71;
        %
        fc(17)=0.8;
        wk(17)=-6.43;
        wd(17)=-0.07;
        wf(17)=-25.51;
        wc(17)=-0.24;
        we(17)=-0.53;
        wj(17)=-6.42;
        wb(17)=-8.38;
        %
        fc(18)=1.;
        wk(18)=-6.33;
        wd(18)=0.10;
        wf(18)=-32.57;
        wc(18)=-0.08;
        we(18)=-1.11;
        wj(18)=-6.30;
        wb(18)=-8.29;
        %
        fc(19)=1.25;
        wk(19)=-6.29;
        wd(19)=0.07;
        wf(19)=-40.02;
        wc(19)=0.00;
        we(19)=-2.25;
        wj(19)=-6.28;
```

```
        wb(19)=-8.27;
        %
        fc(20)=1.6;
        wk(20)=-6.12;
        wd(20)=-0.28;
        wf(20)=-48.47;
        wc(20)=0.06;
        we(20)=-3.99;
        wj(20)=-6.32;
        wb(20)=-8.07;
        %
        fc(21)=2.;
        wk(21)=-5.49;
        wd(21)=-1.01;
        wf(21)=-56.19;
        wc(21)=0.10;
        we(21)=-5.82;
        wj(21)=-6.34;
        wb(21)=-7.60;
        %
        fc(22)=2.5;
        wk(22)=-4.01;
        wd(22)=-2.20;
        wf(22)=-63.93;
        wc(22)=0.15;
        we(22)=-7.77;
        wj(22)=-6.22;
        wb(22)=-6.13;
        %
        fc(23)=3.15;
        wk(23)=-1.90;
        wd(23)=-3.85;
        wf(23)=-71.96;
        wc(23)=0.19;
        we(23)=-9.81;
        wj(23)=-5.62;
        wb(23)=-3.58;
        %
        fc(24)=4.;
        wk(24)=-0.29;
        wd(24)=-5.82;
        wf(24)=-80.26;
        wc(24)=0.20;
        we(24)=-11.93;
        wj(24)=-4.04;
        wb(24)=-1.02;
        %
        fc(25)=5.;
        wk(25)=0.33;
        wd(25)=-7.76;
        wf(25)=-90.;
        wc(25)=0.11;
        we(25)=-13.91;
        wj(25)=-2.01;
        wb(25)=0.21;
        %
        fc(26)=6.3;
        wk(26)=0.46;
```

```
        wd(26)=-9.81;
        wf(26)=-90.;
        wc(26)=-0.23;
        we(26)=-15.94;
        wj(26)=-0.48;
        wb(26)=0.46;
        %
        fc(27)=8.;
        wk(27)=0.31;
        wd(27)=-11.93;
        wf(27)=-90.;
        wc(27)=-1.00;
        we(27)=-18.03;
        wj(27)=0.15;
        wb(27)=0.21;
        %
        fc(28)=10.;
        wk(28)=-0.10;
        wd(28)=-13.91;
        wf(28)=-90.;
        wc(28)=-2.20;
        we(28)=-19.98;
        wj(28)=0.26;
        wb(28)=-0.23;
        %
        fc(29)=12.5;
        wk(29)=-0.89;
        wd(29)=-15.87;
        wf(29)=-90.;
        wc(29)=-3.79;
        we(29)=-21.93;
        wj(29)=0.22;
        wb(29)=-0.85;
        %
        fc(30)=16.;
        wk(30)=-2.28;
        wd(30)=-18.03;
        wf(30)=-90.;
        wc(30)=-5.82;
        we(30)=-24.08;
        wj(30)=0.16;
        wb(30)=-1.83;
        %
        fc(31)=20.;
        wk(31)=-3.93;
        wd(31)=-19.99;
        wf(31)=-90.;
        wc(31)=-7.77;
        we(31)=-26.02;
        wj(31)=0.10;
        wb(31)=-3.00;
        %
        fc(32)=25.;
        wk(32)=-5.80;
        wd(32)=-21.94;
        wf(32)=-90.;
        wc(32)=-9.76;
        we(32)=-27.97;
```

```
        wj(32)=0.06;
        wb(32)=-4.44;
        %
        fc(33)=31.5;
        wk(33)=-7.86;
        wd(33)=-23.98;
        wf(33)=-90.;
        wc(33)=-11.84;
        we(33)=-30.01;
        wj(33)=0.00;
        wb(33)=-6.16;
        %
        fc(34)=40.;
        wk(34)=-10.05;
        wd(34)=-26.13;
        wf(34)=-90.;
        wc(34)=-14.02;
        we(34)=-32.15;
        wj(34)=-0.08;
        wb(34)=-8.11;
        %
        fc(35)=50.;
        wk(35)=-12.19;
        wd(35)=-28.22;
        wf(35)=-90.;
        wc(35)=-16.13;
        we(35)=-34.24;
        wj(35)=-0.24;
        wb(35)=-10.09;
        %
        fc(36)=63.;
        wk(36)=-14.61;
        wd(36)=-30.60;
        wf(36)=-90.;
        wc(36)=-18.53;
        we(36)=-36.62;
        wj(36)=-0.62;
        wb(36)=-12.43;
        %
        fc(37)=80.;
        wk(37)=-17.56;
        wd(37)=-33.53;
        wf(37)=-90.;
        wc(37)=-21.47;
        we(37)=-39.55;
        wj(37)=-1.48;
        wb(37)=-15.34;
        %
        fc(38)=100.;
        wk(38)=-21.04;
        wd(38)=-36.99;
        wf(38)=-90.;
        wc(38)=-24.94;
        we(38)=-43.01;
        wj(38)=-3.01;
        wb(38)=-18.72;
        %
        fc(39)=125.;
```

```
        wk(39)=-25.35;
        wd(39)=-41.28;
        wf(39)=-90.;
        wc(39)=-29.24;
        we(39)=-47.31;
        wj(39)=-5.36;
        wb(39)=-23.00;
        %
        fc(40)=160.;
        wk(40)=-30.91;
        wd(40)=-46.84;
        wf(40)=-90.;
        wc(40)=-34.80;
        we(40)=-52.86;
        wj(40)=-8.78;
        wb(40)=-28.56;
        %
        fc(41)=200.;
        wk(41)=-36.38;
        wd(41)=-52.30;
        wf(41)=-90.;
        wc(41)=-40.26;
        we(41)=-58.33;
        wj(41)=-12.30;
        wb(41)=-34.03;
        %
        fc(42)=250.;
        wk(42)=-42.0;
        wd(42)=-57.97;
        wf(42)=-90.;
        wc(42)=-45.92;
        we(42)=-63.99;
        wj(42)=-16.03;
        wb(42)=-39.69;
        %
        fc(43)=315.;
        wk(43)=-48.00;
        wd(43)=-63.92;
        wf(43)=-90.;
        wc(43)=-51.88;
        we(43)=-69.94;
        wj(43)=-19.98;
        wb(43)=-46.65;
        %
        fc(44)=400.;
        wk(44)=-54.20;
        wd(44)=-70.12;
        wf(44)=-90.;
        wc(44)=-58.08;
        we(44)=-76.14;
        wj(44)=-24.10;
        wb(44)=-51.84;
        %
        for i=1:44
            %
            wk(i)=10^(wk(i)/20.);
            wd(i)=10^(wd(i)/20.);
            wf(i)=10^(wf(i)/20.);
```

```
            wc(i)=10^(wc(i)/20.);
            we(i)=10^(we(i)/20.);
            wj(i)=10^(wj(i)/20.);
            wb(i)=10^(wb(i)/20.);
            %
        end
        %
        www=zeros(44,1);
        %
        for i=1:44
            %
            if(iweight==1)
                www(i)=wk(i);
            end
            if(iweight==2)
                www(i)=wd(i);
            end
            if(iweight==3)
                www(i)=wf(i);
            end
            if(iweight==4)
                www(i)=wc(i);
            end
            if(iweight==5)
                www(i)=we(i);
            end
            if(iweight==6)
                www(i)=wj(i);
            end
            if(iweight==7)
                www(i)=wb(i);
            end
            %
        end
        %
        imax=length(fc);
        %
        fw=fc;
        %
        fwl=fw/(2^(1/6));
        %
        fwu=zeros(imax,1);
        %
        for i=1:imax-1
            fwu(i)=fwl(i+1);
        end
        %
        fwu(imax)=fw(imax)*(2^(1/6));
        aw=zeros(length(amp),1);
        %
        iband=3;  % bandpass filtering
        iphase=1; % refiltering for phase correction
        %
        progressbar;
        for i=1:44
            %
            progressbar(i/44);
            %
```

```
            fh=fwl(i);  % highpass filter frequency
            fl=fwu(i);  % lowpass filter frequency
            %
            if(fl<sr/2.1)
                [y,mu,sd,rms(i)]=...
                    Butterworth_filter_function_alt(amp,dt,iband,fl,fh,iphase);
                %
                aw=aw+y*www(i);
            end
            %
        end
        %
        pause(0.5);
        progressbar(1);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %
        n=length(aw);
        %
        AW=std(aw);
        AU=std(amp);
        MTW=AW;
        MTU=AU;
        %
        VDVu=0;
        VDVu_run=zeros(n,1);
        for i=1:n
            VDVu=VDVu+amp(i)^4;
            VDVu_run(i,1)=(VDVu*dt)^(0.25);
        end
        VDVu=(VDVu*dt)^0.25;
        %VDVu_run=transpose(VDVu_run);
        %
        VDVw=0;
        VDVw_run=zeros(n,1);
        for i=1:n
            VDVw=VDVw+aw(i)^4;
            VDVw_run(i,1)=(VDVw*dt)^0.25;
        end
        VDVw=(VDVw*dt)^0.25;
        %VDVw_run=transpose(VDVw_run);
        %
        aurms=0;
        aurms_run=zeros(n,1);
        for i=1:n
            aurms=aurms+amp(i)^2;
            aurms_run(i,1)=(aurms*dt/tim(i))^0.5;
        end
        aurms=aurms_run(n);
        %aurms_run=transpose(aurms_run);
        %
        awrms=0;
        awrms_run=zeros(n,1);
        for i=1:n
            awrms=awrms+aw(i)^2;
            awrms_run(i,1)=(awrms*dt/tim(i))^0.5;
        end
        awrms=awrms_run(n);
        %awrms_run=awrms_run.';
```

```
        %
        %disp(' ');
        %disp(' Subdivide time history into segments?  1=yes 2=no');
        %ig=input(' ');
        ig=1;
        %
        if(ig==1)
            %disp(' ');
            %dur=input(' Enter segment duration (sec) ');
            dur=1;
            %
            ns=round(dur/dt);
            %
            loops=floor(n/ns);
            %
            ia=1;
            %
            disp(' ');
            %disp(' Time          aw      ');
            %disp(' (sec)      (m/sec^2) RMS ');
            %
            for i=1:loops
                ib=ia+ns-1;
                if(ib>n)
                    break;
                end
                ttt=dt*(ib+ia)/2;
                ttt_run(i)=ttt;
                asu=std(amp(ia:ib));
                asu_run(i)=asu;
                out1=sprintf('%8.0f  %8.2f ',ttt,asu);
                %disp(out1);
                if(asu>MTU)
                    MTU=asu;
                end
                ia=ib;
            end
            %
            ia=1;
            %
            for i=1:loops
                ib=ia+ns-1;
                if(ib>n)
                    break;
                end
                ttt=dt*(ib+ia)/2;
                ttt_run(i)=ttt;
                asw=std(aw(ia:ib));
                asw_run(i)=asw;
                out1=sprintf('%8.0f  %8.2f ',ttt,asw);
                %disp(out1);
                if(asw>MTW)
                    MTW=asw;
                end
                ia=ib;
            end
            %
        end
```

```
        disp(' ');

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %
        disp(name{m})

        %
        out1=sprintf('\n Unweighted R.M.S. Acceleration  aurms = %8.4g (m/sec^2)RMS
',aurms);
        disp(out1);
        %
        out1=sprintf('\n Unweighted Fourth Power Vibration Dose VDVu = %8.4g (m/
sec^(1.75)) \n',VDVu);
        disp(out1);
        %
        %
        out1=sprintf('\n Peak Weighted Acceleration  max(aw(t)) = %8.4g m/sec^2
',max(aw));
        disp(out1);
        %
        out1=sprintf('\n Weighted R.M.S. Acceleration  awrms = %8.4g (m/sec^2)RMS
',awrms);
        disp(out1);
        %
        out1=sprintf('\n Weighted Fourth Power Vibration Dose VDVw = %8.4g (m/
sec^(1.75)) \n',VDVw);
        disp(out1);
        %
        out1=sprintf('\n                          CF = %8.4g\n',max(aw)/awrms);
        disp(out1);
```

```
Back Plate: X-Axis

 Unweighted R.M.S. Acceleration  aurms =   0.9199 (m/sec^2)RMS

 Unweighted Fourth Power Vibration Dose VDVu =    10.67 (m/sec^(1.75))


 Peak Weighted Acceleration  max(aw(t)) =    9.945 m/sec^2

 Weighted R.M.S. Acceleration  awrms =   0.6691 (m/sec^2)RMS

 Weighted Fourth Power Vibration Dose VDVw =    7.415 (m/sec^(1.75))


                         CF =    14.86

Back Plate: Y-Axis

 Unweighted R.M.S. Acceleration  aurms =   0.8077 (m/sec^2)RMS

 Unweighted Fourth Power Vibration Dose VDVu =    11.54 (m/sec^(1.75))


 Peak Weighted Acceleration  max(aw(t)) =    6.531 m/sec^2

 Weighted R.M.S. Acceleration  awrms =   0.3334 (m/sec^2)RMS

 Weighted Fourth Power Vibration Dose VDVw =    3.966 (m/sec^(1.75))
```

CF = 19.59

Back Plate: Z-Axis

Unweighted R.M.S. Acceleration aurms = 0.5189 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 8.08 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 3.518 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.3026 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 3.747 (m/sec^(1.75))

CF = 11.63

Chassis Frame: X-Axis

Unweighted R.M.S. Acceleration aurms = 0.5021 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 4.81 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 3.728 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.2821 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 2.855 (m/sec^(1.75))

CF = 13.21

Chassis Frame: Y-Axis

Unweighted R.M.S. Acceleration aurms = 0.4418 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 6.472 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 3.773 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.2141 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 2.607 (m/sec^(1.75))

CF = 17.63

Chassis Frame: Z-Axis

Unweighted R.M.S. Acceleration aurms = 0.5352 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 8.23 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 7.129 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.4737 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 7.496 (m/sec^(1.75))

CF = 15.05

Seat Pan: X-Axis

Unweighted R.M.S. Acceleration aurms = 0.5 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 4.708 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 3.621 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.2913 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 2.928 (m/sec^(1.75))

CF = 12.43

Seat Pan: Y-Axis

Unweighted R.M.S. Acceleration aurms = 0.3592 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 4.992 (m/sec^(1.75))

Peak Weighted Acceleration max(aw(t)) = 2.044 m/sec^2

Weighted R.M.S. Acceleration awrms = 0.1361 (m/sec^2)RMS

Weighted Fourth Power Vibration Dose VDVw = 1.344 (m/sec^(1.75))

CF = 15.01

Seat Pan: Z-Axis

Unweighted R.M.S. Acceleration aurms = 0.4162 (m/sec^2)RMS

Unweighted Fourth Power Vibration Dose VDVu = 5.653 (m/sec^(1.75))

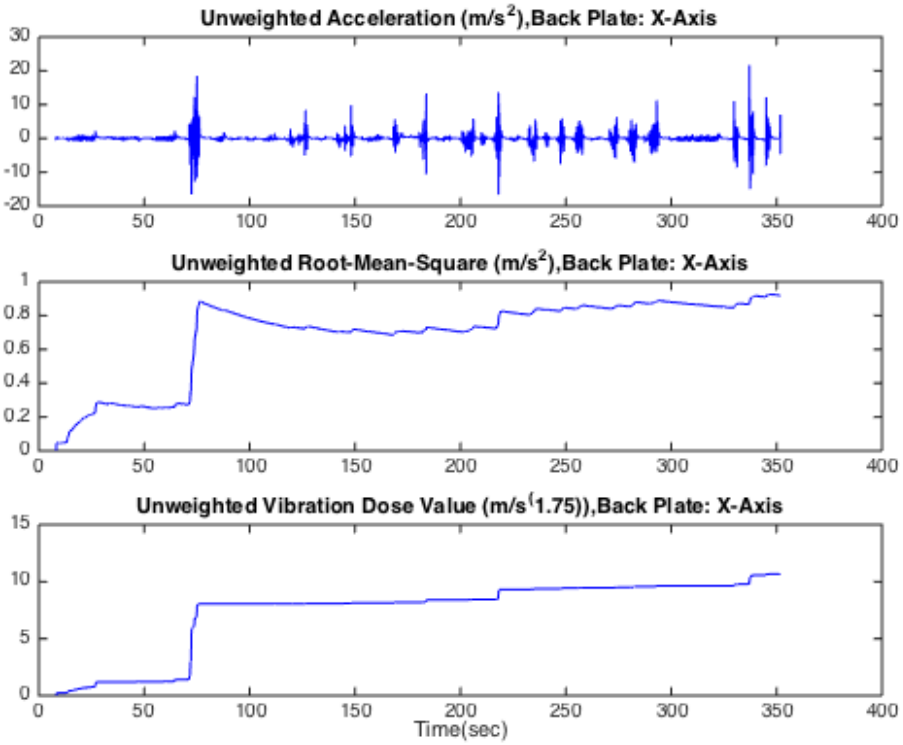
Peak Weighted Acceleration max(aw(t)) = 4.974 m/sec^2

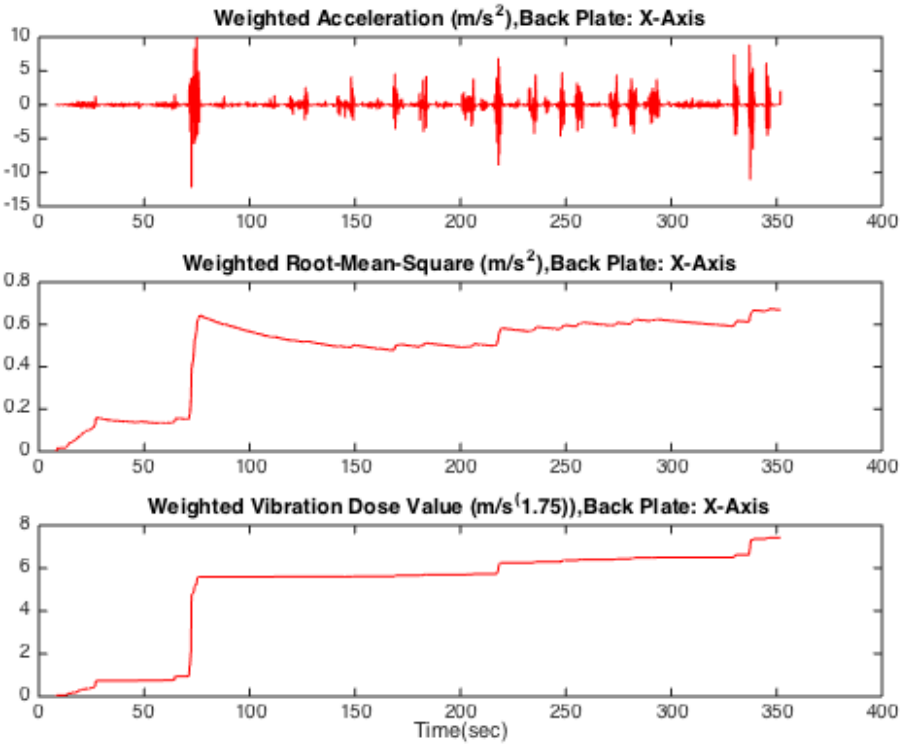
Weighted R.M.S. Acceleration awrms = 0.3673 (m/sec^2)RMS

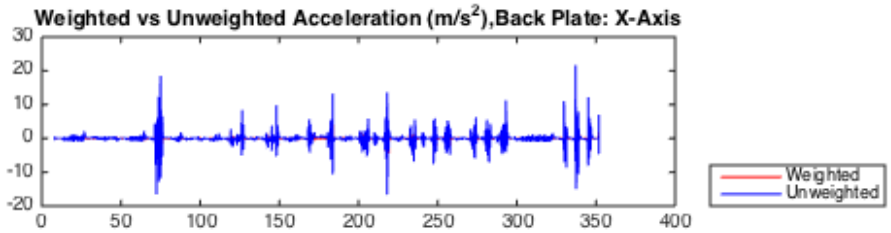
Weighted Fourth Power Vibration Dose VDVw = 4.773 (m/sec^(1.75))

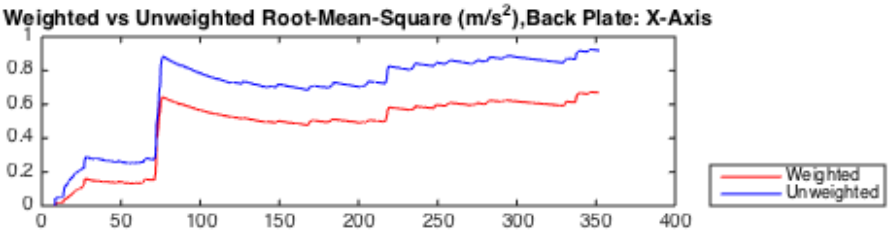
CF = 13.54

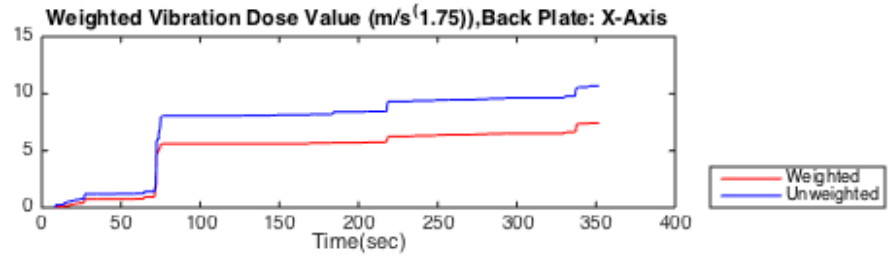
```
        figure
        title(['Unweighted Vibration Analysis,',name{m}])
        subplot (3,1,1)
        plot(tim,amp,'b')
        title(['Unweighted Acceleration (m/s^2),',name{m}])

        subplot (3,1,2)
        plot(tim,aurms_run,'b')
        title(['Unweighted Root-Mean-Square (m/s^2),',name{m}])

        subplot (3,1,3)
        plot(tim,VDVu_run,'b')
        title(['Unweighted Vibration Dose Value (m/s^(1.75)),',name{m}])
        xlabel('Time(sec)');
        %
        figure
        title(['Weighted Vibration Analysis,',name{m}])
        subplot (3,1,1)
        plot(tim,aw,'r')
        title(['Weighted Acceleration (m/s^2),',name{m}])

        subplot (3,1,2)
        plot(tim,awrms_run,'r')
        title(['Weighted Root-Mean-Square (m/s^2),',name{m}])

        subplot (3,1,3)
        plot(tim,VDVw_run,'r')
        title(['Weighted Vibration Dose Value (m/s^(1.75)),',name{m}])
        xlabel('Time(sec)');
        %
        %
        figure
        title(['Weighted vs Unweighted Vibration Analysis,',name{m}])
        subplot (3,1,1)
        plot(tim,aw,'r',tim,amp,'b')
        title(['Weighted vs Unweighted Acceleration (m/s^2),',name{m}])
        legend('Weighted','Unweighted','Location','SouthEastOutside')

        subplot (3,1,2)
        plot(tim,awrms_run,'r',tim,aurms_run,'b')
        title(['Weighted vs Unweighted Root-Mean-Square (m/s^2),',name{m}])
        legend('Weighted','Unweighted','Location','SouthEastOutside')

        subplot (3,1,3)
        plot(tim,VDVw_run,'r',tim,VDVu_run,'b')
        title(['Weighted Vibration Dose Value (m/s^(1.75)),',name{m}])
        legend('Weighted','Unweighted','Location','SouthEastOutside')
        xlabel('Time(sec)');
```

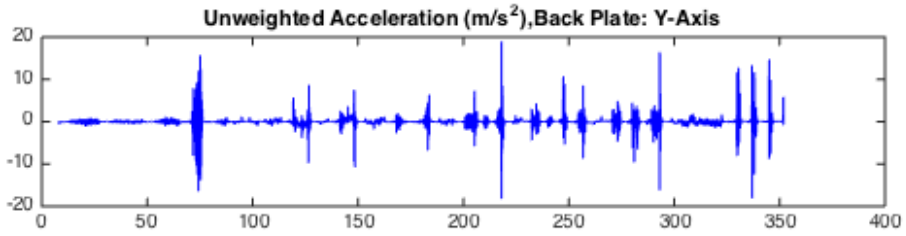

Unweighted Acceleration (m/s^2),Back Plate: X-Axis
Unweighted Root-Mean-Square (m/s^2),Back Plate: X-Axis
Unweighted Vibration Dose Value (m/s^1.75)),Back Plate: X-Axis
Time(sec)

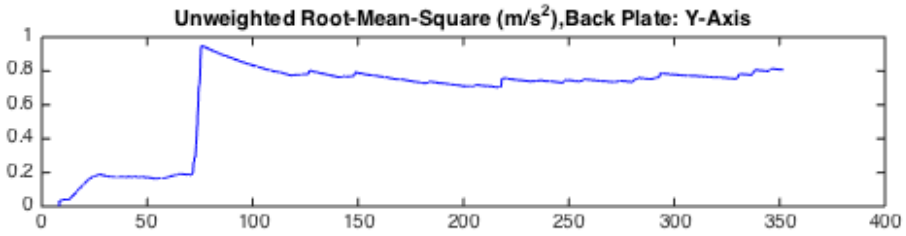

Weighted Acceleration (m/s^2),Back Plate: X-Axis
Weighted Root-Mean-Square (m/s^2),Back Plate: X-Axis
Weighted Vibration Dose Value (m/s^1.75)),Back Plate: X-Axis
Time(sec)

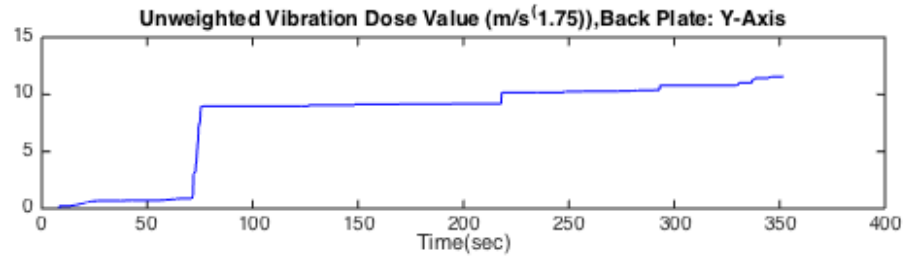

Weighted vs Unweighted Acceleration (m/s2),Back Plate: X-Axis
Weighted
Unweighted

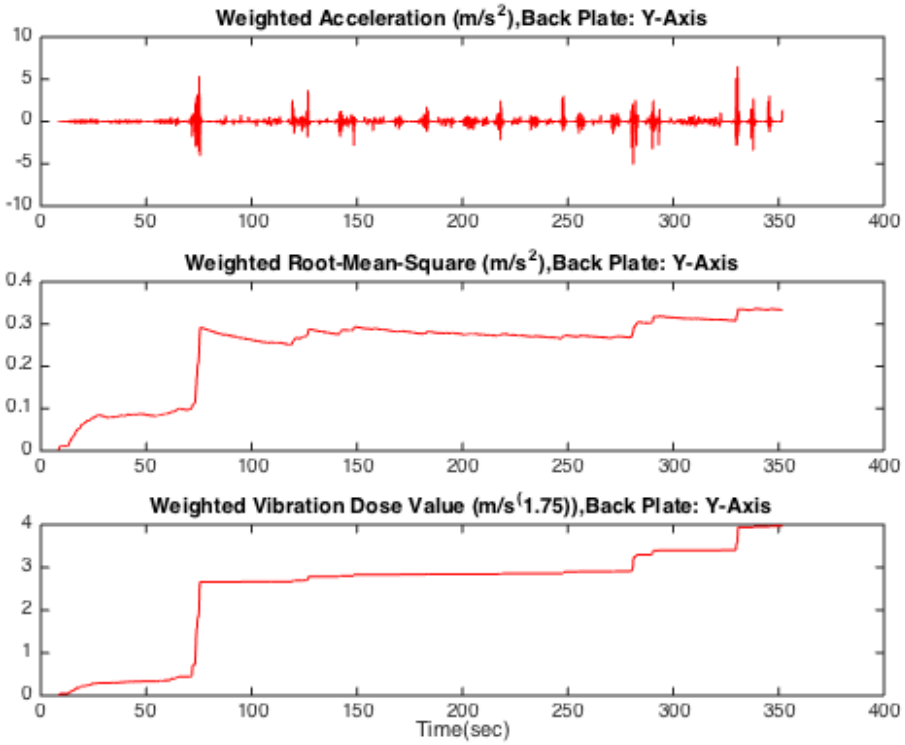

Weighted vs Unweighted Root-Mean-Square (m/s2),Back Plate: X-Axis
Weighted
Unweighted

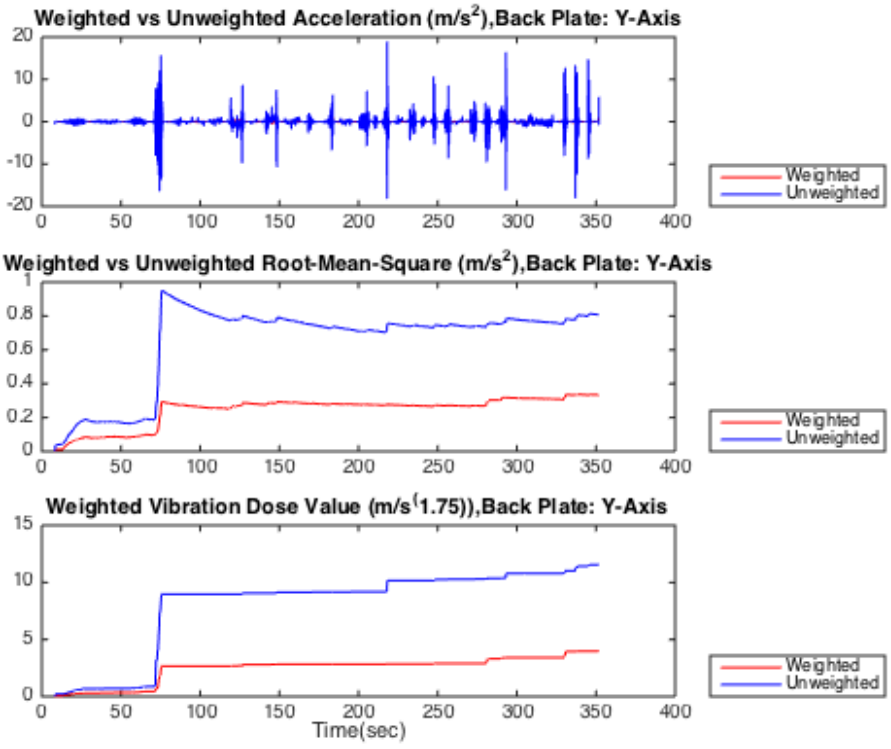

Weighted Vibration Dose Value (m/s(1.75)),Back Plate: X-Axis
Weighted
Unweighted
Time(sec)

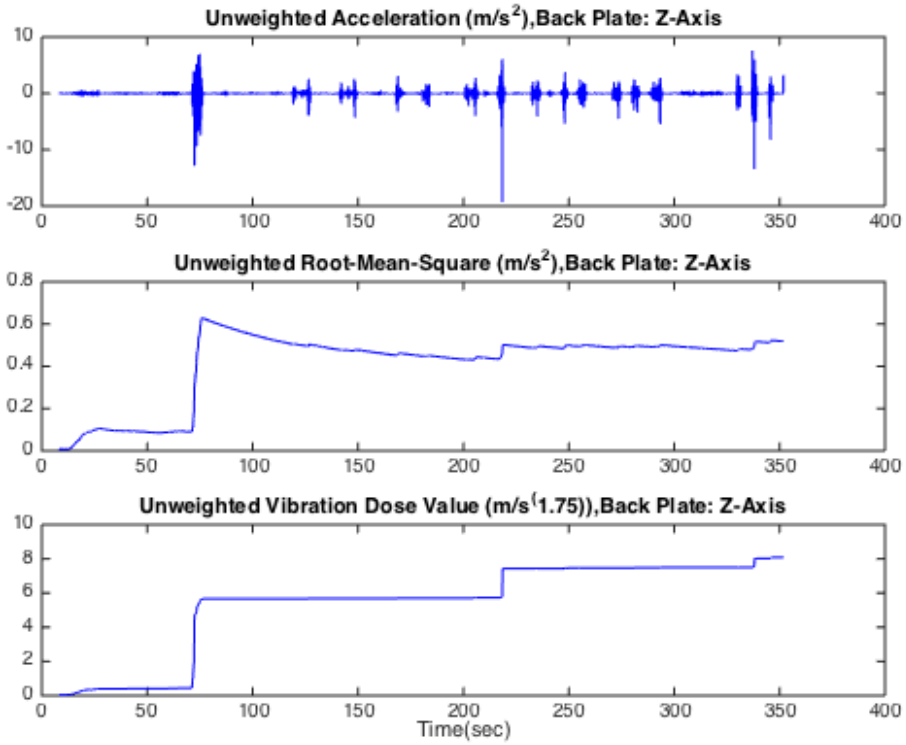

Unweighted Acceleration (m/s2),Back Plate: Y-Axis

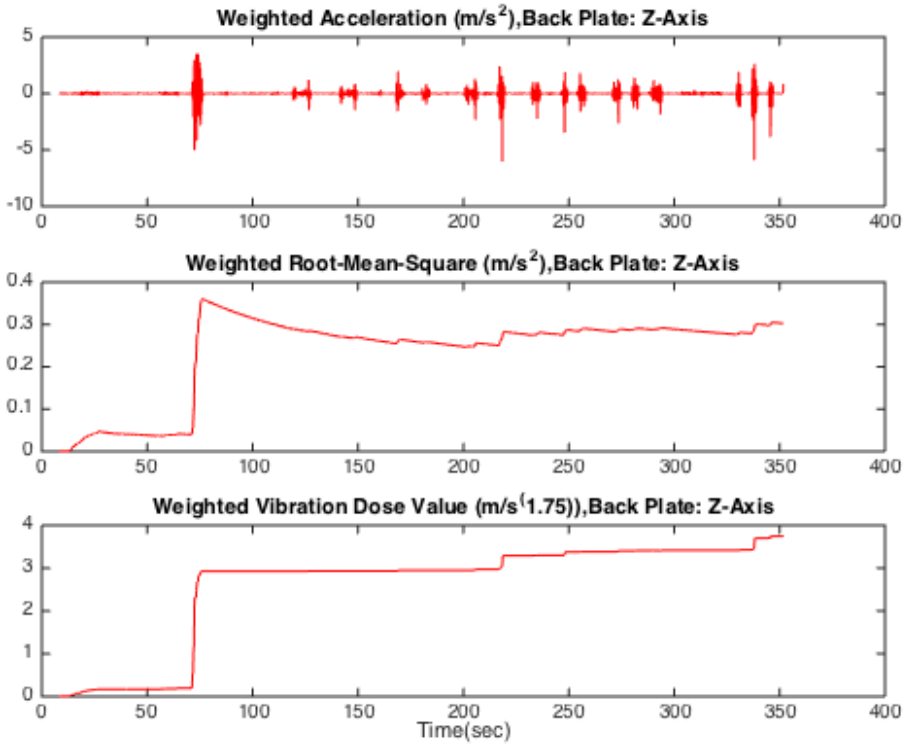

Unweighted Root-Mean-Square (m/s2),Back Plate: Y-Axis

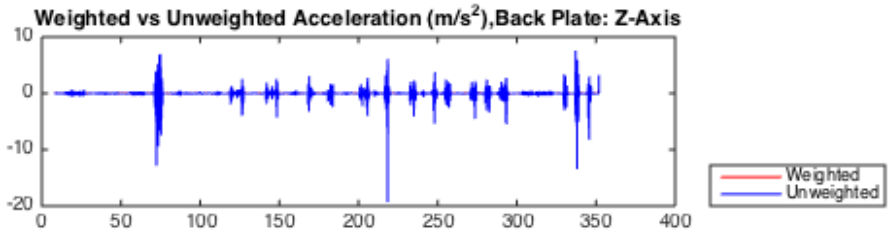

Unweighted Vibration Dose Value (m/s(1.75)),Back Plate: Y-Axis
Time(sec)

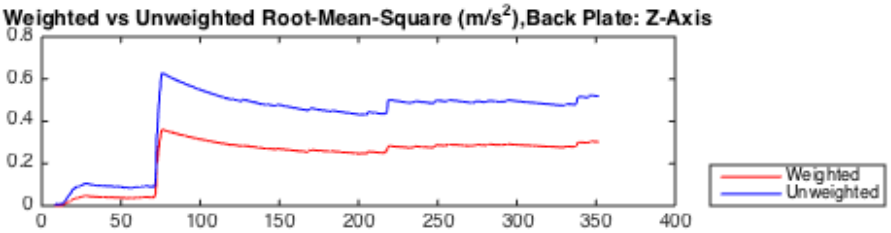

Weighted Acceleration (m/s2),Back Plate: Y-Axis
Weighted Root-Mean-Square (m/s2),Back Plate: Y-Axis
Weighted Vibration Dose Value (m/s1.75)),Back Plate: Y-Axis
Time(sec)

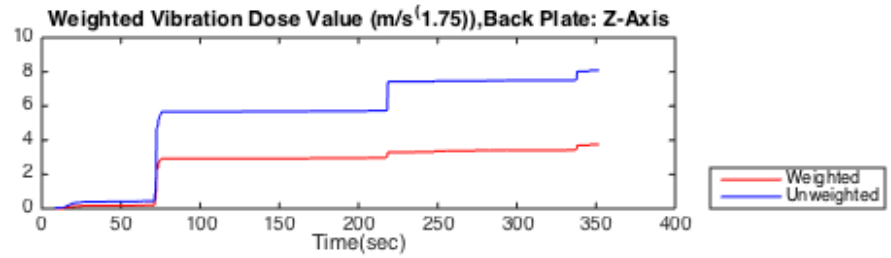

Weighted vs Unweighted Acceleration (m/s2),Back Plate: Y-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s2),Back Plate: Y-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s1.75)),Back Plate: Y-Axis
Weighted
Unweighted
Time(sec)

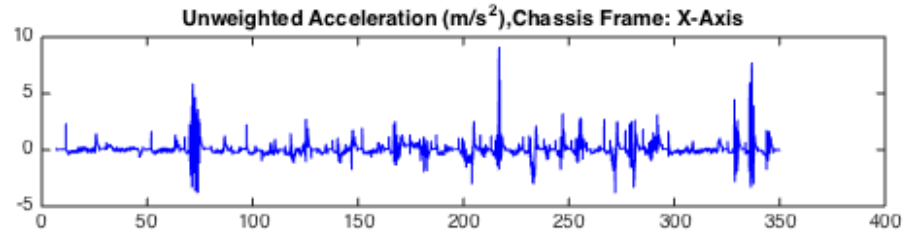

Unweighted Acceleration (m/s^2),Back Plate: Z-Axis
Unweighted Root-Mean-Square (m/s^2),Back Plate: Z-Axis
Unweighted Vibration Dose Value (m/s^1.75)),Back Plate: Z-Axis
Time(sec)

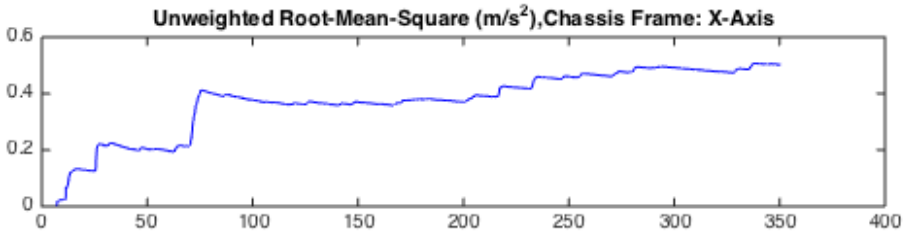

Weighted Acceleration (m/s^2),Back Plate: Z-Axis
Weighted Root-Mean-Square (m/s^2),Back Plate: Z-Axis
Weighted Vibration Dose Value (m/s^1.75)),Back Plate: Z-Axis
Time(sec)

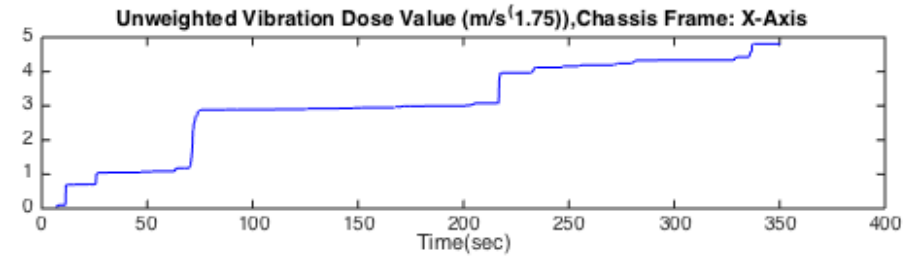

Weighted vs Unweighted Acceleration (m/s2),Back Plate: Z-Axis
Weighted
Unweighted

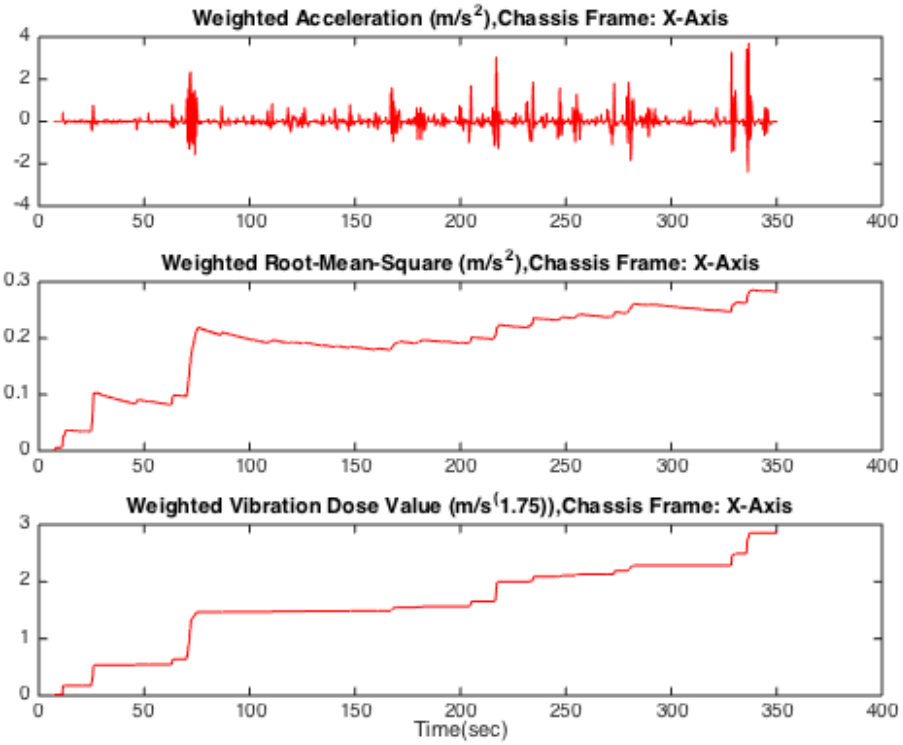

Weighted vs Unweighted Root-Mean-Square (m/s2),Back Plate: Z-Axis
Weighted
Unweighted

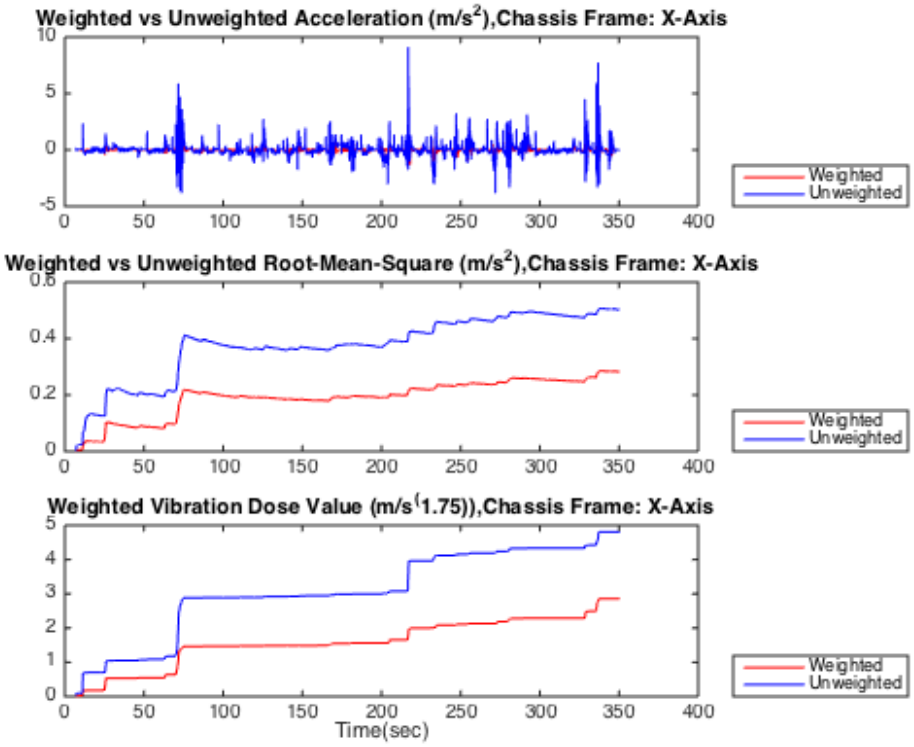

Weighted Vibration Dose Value (m/s1.75)),Back Plate: Z-Axis
Weighted
Unweighted
Time(sec)

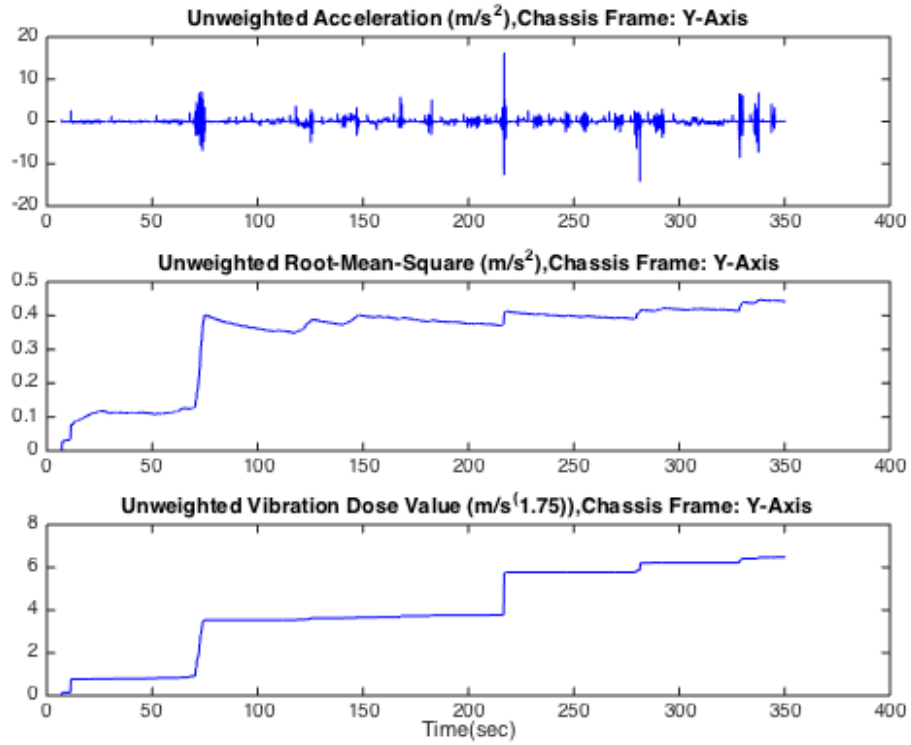

Unweighted Acceleration (m/s2),Chassis Frame: X-Axis

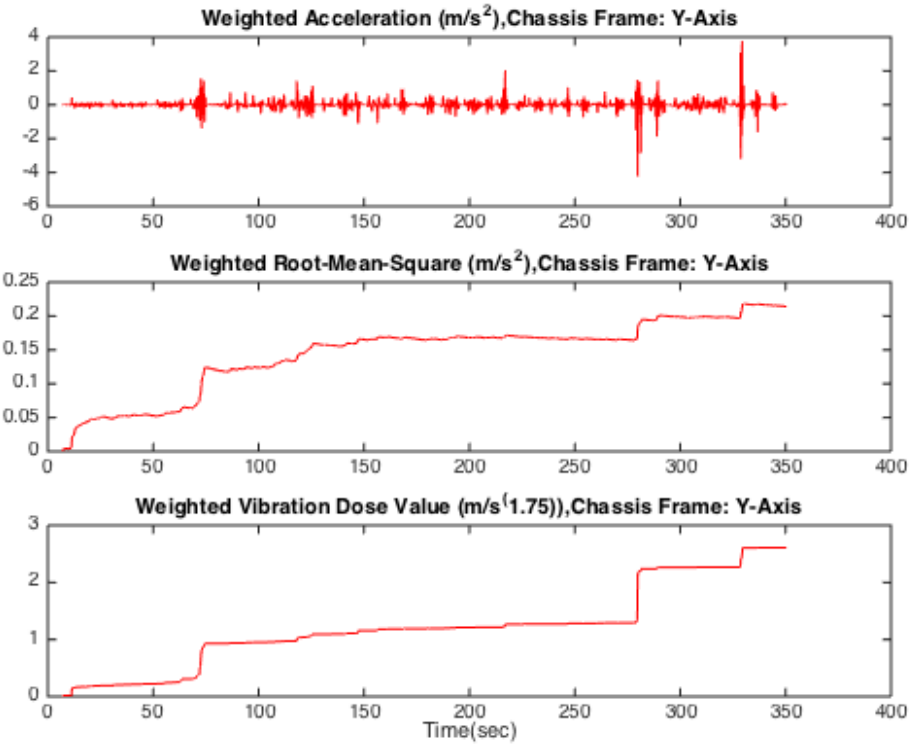

Unweighted Root-Mean-Square (m/s2),Chassis Frame: X-Axis

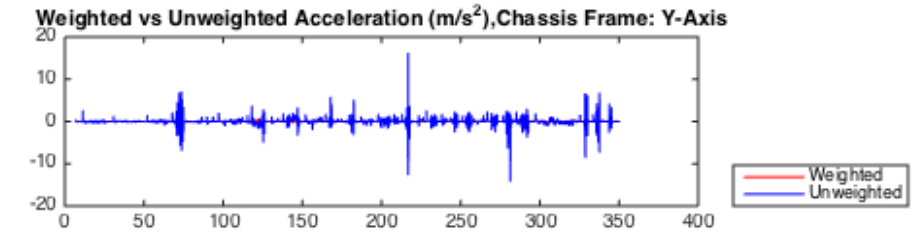

Unweighted Vibration Dose Value (m/s1.75)),Chassis Frame: X-Axis
Time(sec)

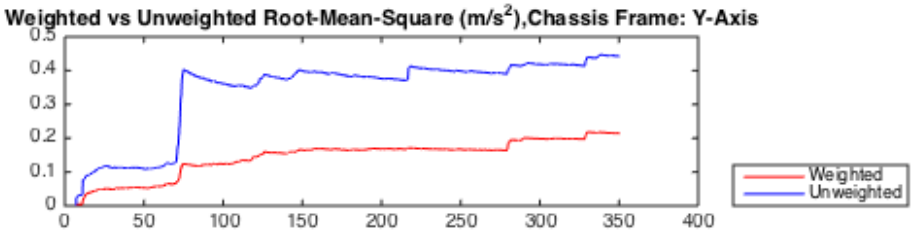

Weighted Acceleration (m/s^2),Chassis Frame: X-Axis
Weighted Root-Mean-Square (m/s^2),Chassis Frame: X-Axis
Weighted Vibration Dose Value (m/s$^{(1.75)}$),Chassis Frame: X-Axis
Time(sec)

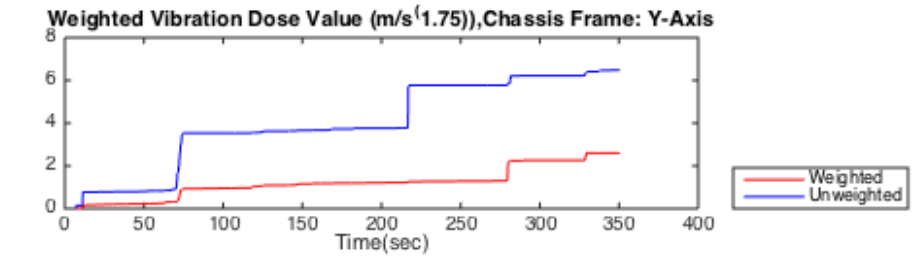

Weighted vs Unweighted Acceleration (m/s^2),Chassis Frame: X-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s^2),Chassis Frame: X-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s$^{(1.75)}$),Chassis Frame: X-Axis
Weighted
Unweighted
Time(sec)

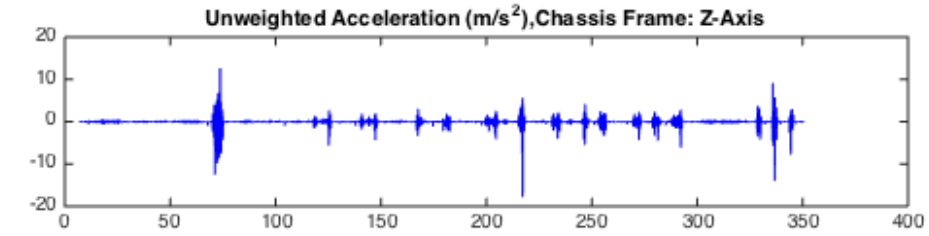

Unweighted Acceleration (m/s^2),Chassis Frame: Y-Axis
Unweighted Root-Mean-Square (m/s^2),Chassis Frame: Y-Axis
Unweighted Vibration Dose Value (m/s^1.75)),Chassis Frame: Y-Axis
Time(sec)

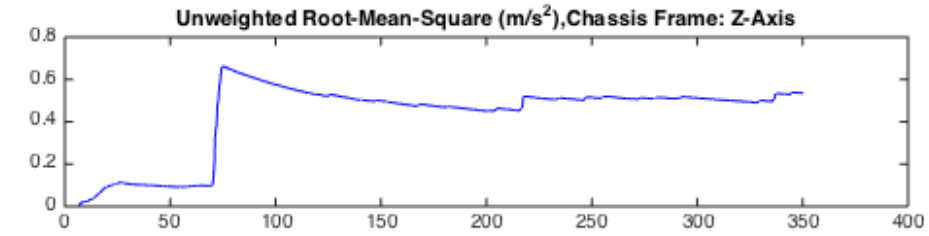

Weighted Acceleration (m/s^2),Chassis Frame: Y-Axis
Weighted Root-Mean-Square (m/s^2),Chassis Frame: Y-Axis
Weighted Vibration Dose Value (m/s^1.75)),Chassis Frame: Y-Axis
Time(sec)

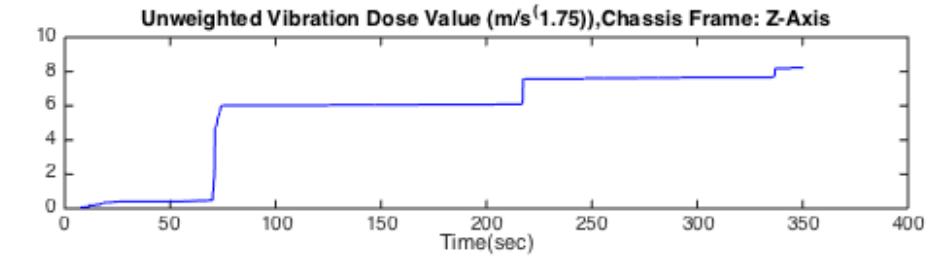

Weighted vs Unweighted Acceleration (m/s2),Chassis Frame: Y-Axis
Weighted
Unweighted

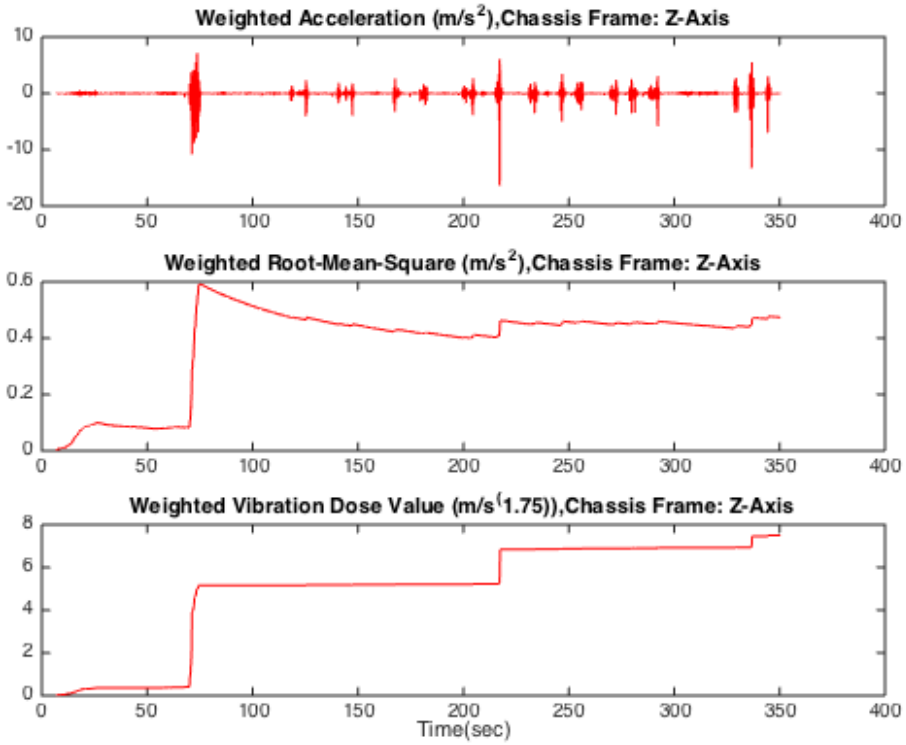

Weighted vs Unweighted Root-Mean-Square (m/s2),Chassis Frame: Y-Axis
Weighted
Unweighted

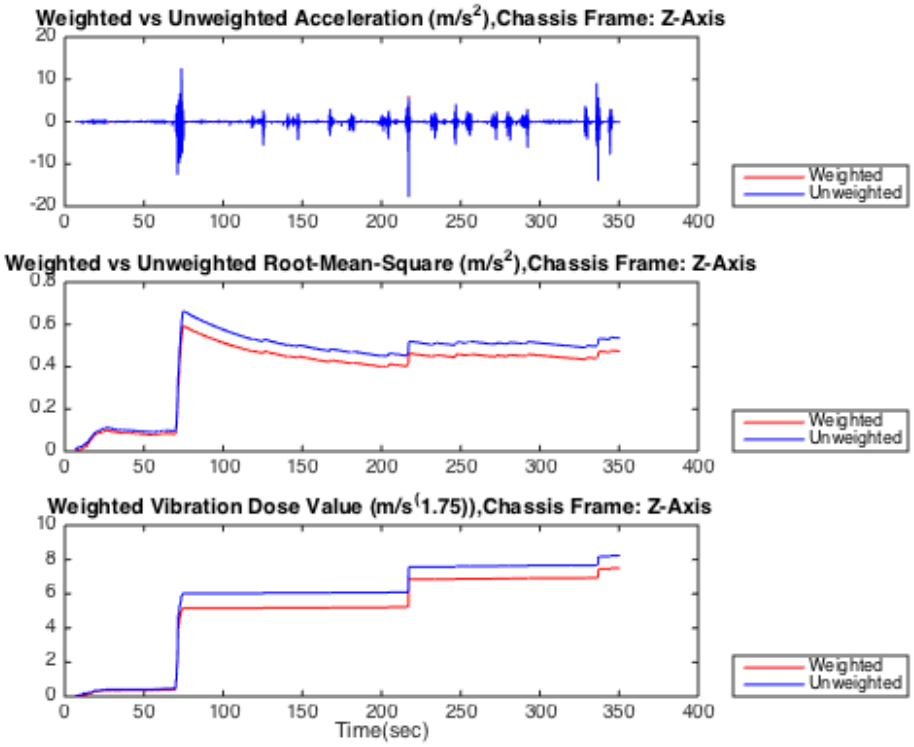

Weighted Vibration Dose Value (m/s1.75)),Chassis Frame: Y-Axis
Weighted
Unweighted
Time(sec)

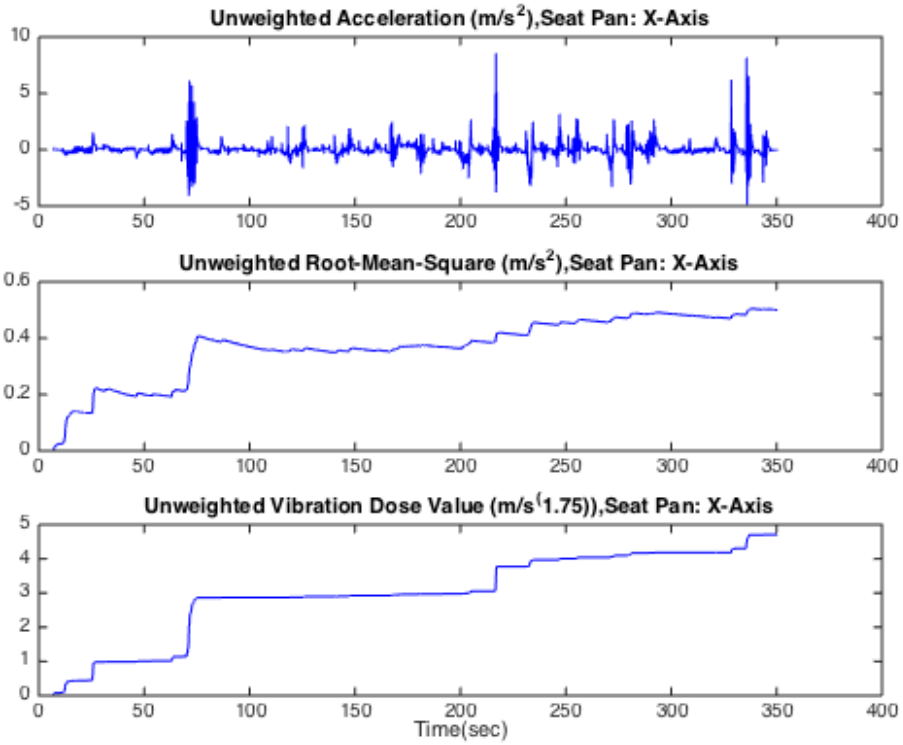

Unweighted Acceleration (m/s2),Chassis Frame: Z-Axis

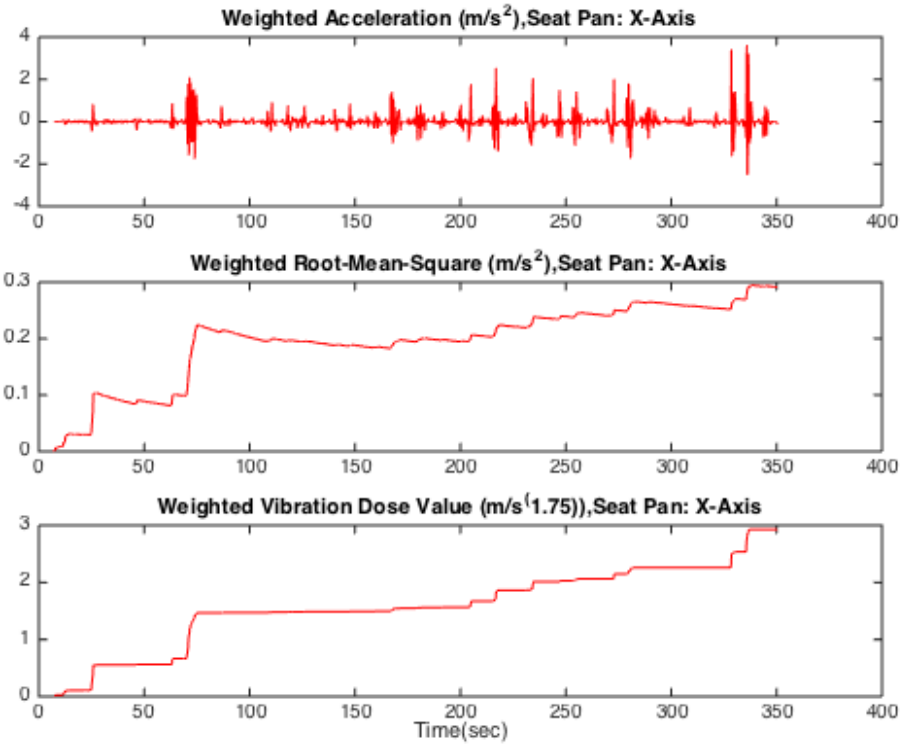

Unweighted Root-Mean-Square (m/s2),Chassis Frame: Z-Axis

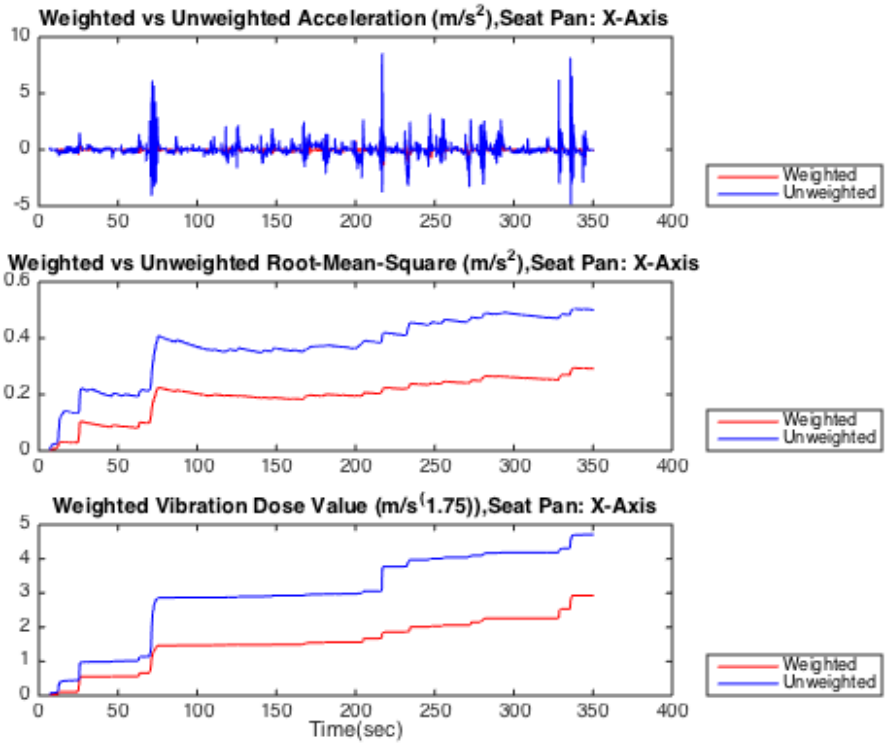

Unweighted Vibration Dose Value (m/s1.75)),Chassis Frame: Z-Axis
Time(sec)

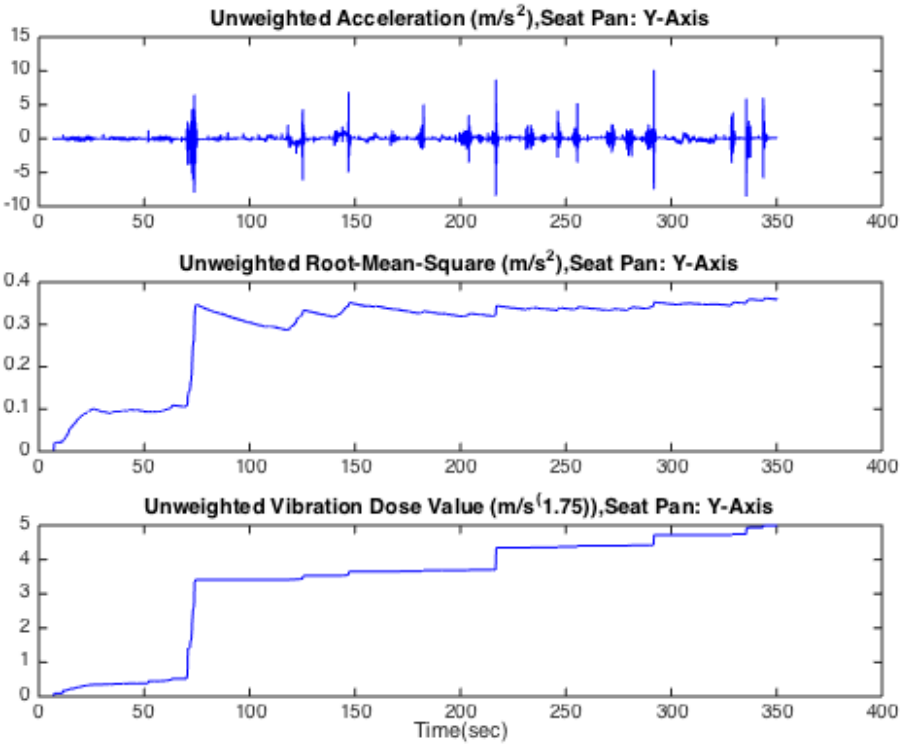

Weighted Acceleration (m/s^2),Chassis Frame: Z-Axis
Weighted Root-Mean-Square (m/s^2),Chassis Frame: Z-Axis
Weighted Vibration Dose Value (m/s^(1.75)),Chassis Frame: Z-Axis
Time(sec)

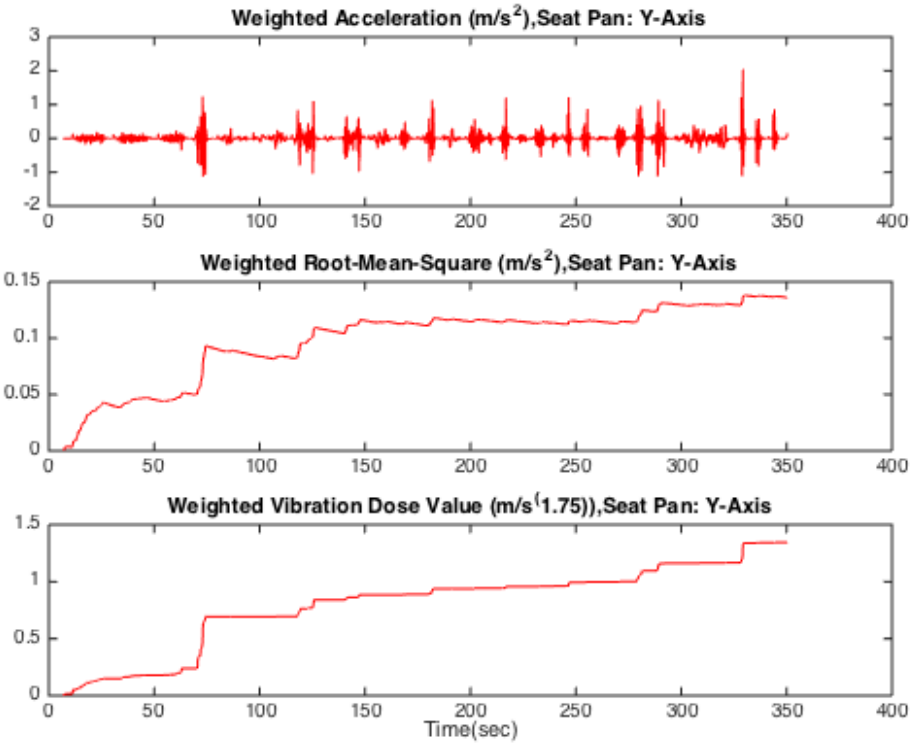

Weighted vs Unweighted Acceleration (m/s^2),Chassis Frame: Z-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s^2),Chassis Frame: Z-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s^(1.75)),Chassis Frame: Z-Axis
Weighted
Unweighted
Time(sec)

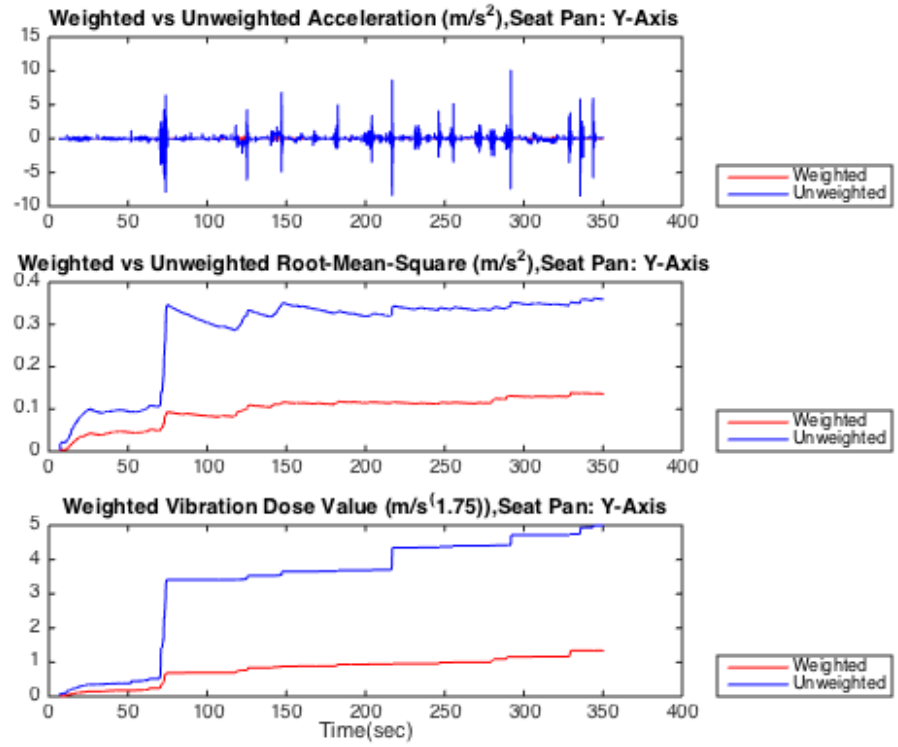

Unweighted Acceleration (m/s2),Seat Pan: X-Axis
Unweighted Root-Mean-Square (m/s2),Seat Pan: X-Axis
Unweighted Vibration Dose Value (m/s(1.75)),Seat Pan: X-Axis
Time(sec)

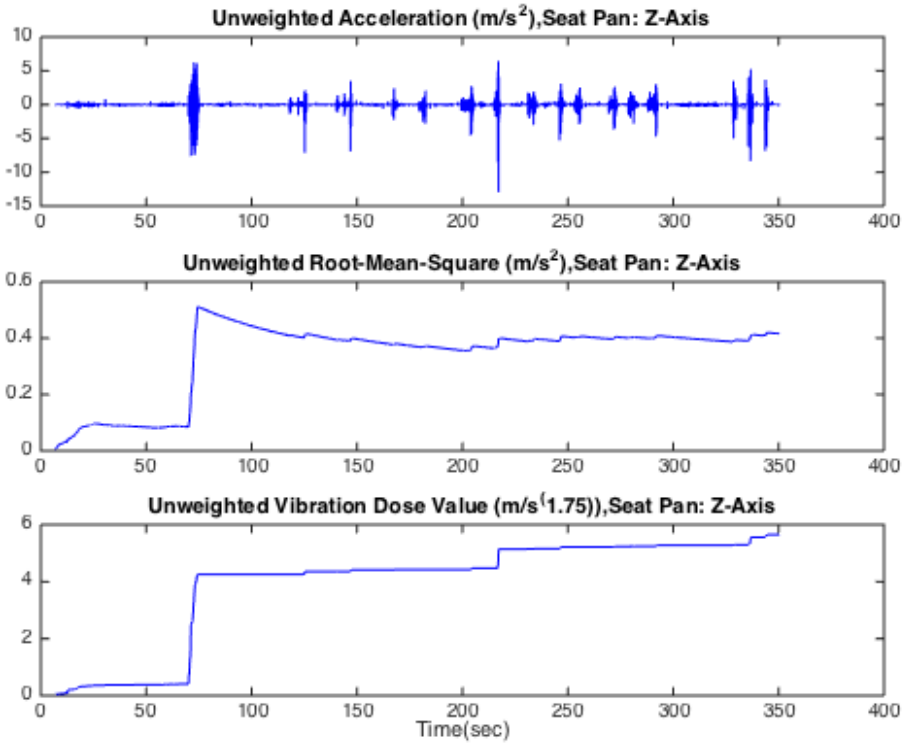

Weighted Acceleration (m/s2),Seat Pan: X-Axis
Weighted Root-Mean-Square (m/s2),Seat Pan: X-Axis
Weighted Vibration Dose Value (m/s(1.75)),Seat Pan: X-Axis
Time(sec)

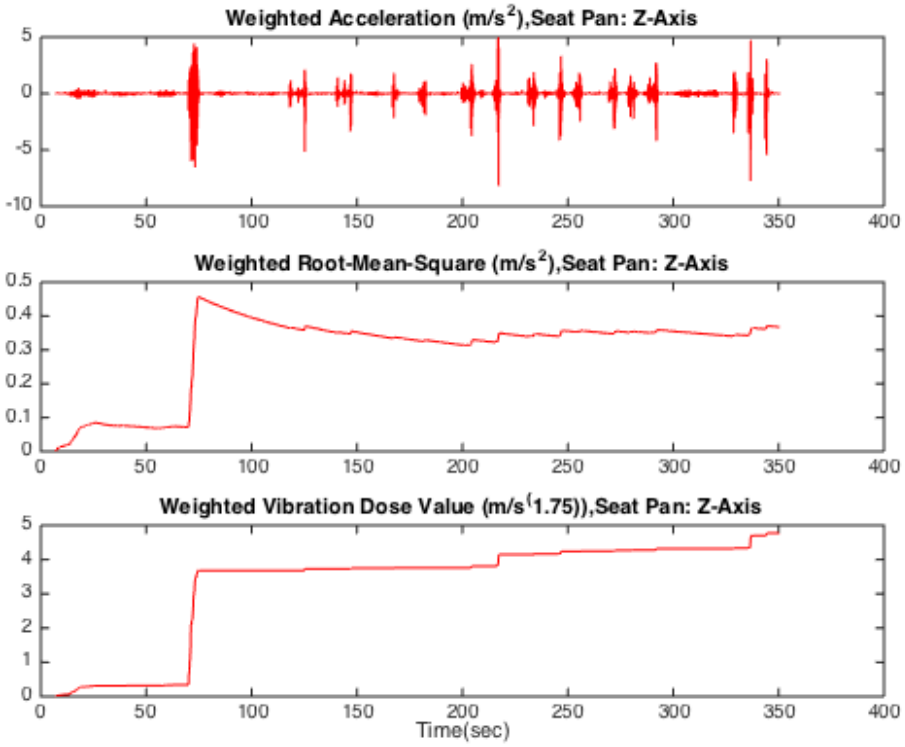

Weighted vs Unweighted Acceleration (m/s2),Seat Pan: X-Axis
Weighted
Unweighted
Weighted vs Unweighted Root-Mean-Square (m/s2),Seat Pan: X-Axis
Weighted
Unweighted
Weighted Vibration Dose Value (m/s1.75)),Seat Pan: X-Axis
Weighted
Unweighted
Time(sec)

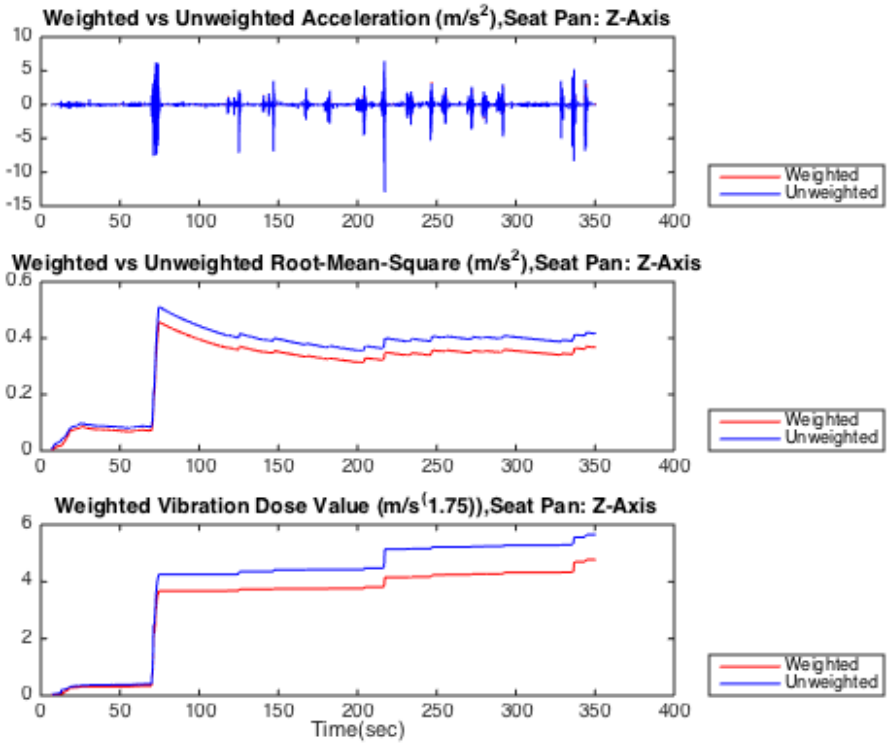

Unweighted Acceleration (m/s2),Seat Pan: Y-Axis
Unweighted Root-Mean-Square (m/s2),Seat Pan: Y-Axis
Unweighted Vibration Dose Value (m/s1.75)),Seat Pan: Y-Axis
Time(sec)


Weighted Acceleration (m/s2),Seat Pan: Y-Axis
Weighted Root-Mean-Square (m/s2),Seat Pan: Y-Axis
Weighted Vibration Dose Value (m/s(1.75)),Seat Pan: Y-Axis
Time(sec)


Weighted vs Unweighted Acceleration (m/s2),Seat Pan: Y-Axis
Weighted vs Unweighted Root-Mean-Square (m/s2),Seat Pan: Y-Axis
Weighted Vibration Dose Value (m/s(1.75)),Seat Pan: Y-Axis
Time(sec)
Weighted
Unweighted


Unweighted Acceleration (m/s2),Seat Pan: Z-Axis
Unweighted Root-Mean-Square (m/s2),Seat Pan: Z-Axis
Unweighted Vibration Dose Value (m/s(1.75)),Seat Pan: Z-Axis
Time(sec)


Weighted Acceleration (m/s2),Seat Pan: Z-Axis
Weighted Root-Mean-Square (m/s2),Seat Pan: Z-Axis
Weighted Vibration Dose Value (m/s(1.75)),Seat Pan: Z-Axis
Time(sec)




```
        if a==1 && m==1
            tim_bx = tim;
            amp_bx=amp;
            aurms_run_bx=aurms_run;
            VDVu_run_bx=VDVu_run;
            aw_bx=aw;
            awrms_run_bx=awrms_run;
            awrms_bx=awrms;
            VDVw_run_bx=VDVw_run;
            VDVw_bx=VDVw;
        elseif a==1 && m==2
            tim_by = tim;
            amp_by=amp;
            aurms_run_by=aurms_run;
            VDVu_run_by=VDVu_run;
            aw_by=aw;
            awrms_run_by=awrms_run;
            awrms_by=awrms;
            VDVw_run_by=VDVw_run;
            VDVw_by=VDVw;
        elseif a==1 && m==3
            tim_bz = tim;
            amp_bz=amp;
            aurms_run_bz=aurms_run;
            VDVu_run_bz=VDVu_run;
            aw_bz=aw;
            awrms_run_bz=awrms_run;
            awrms_bz=awrms;
            VDVw_run_bz=VDVw_run;
            VDVw_bz=VDVw;
```

```
        elseif a==2 && m==1
            tim_fx = tim;
            amp_bx=amp;
            aurms_run_fx=aurms_run;
            VDVu_run_fx=VDVu_run;
            aw_fx=aw;
            awrms_run_fx=awrms_run;
            awrms_fx=awrms;
            VDVw_run_fx=VDVw_run;
            VDVw_fx=VDVw;
        elseif a==2 && m==2
            tim_fy = tim;
            amp_fy=amp;
            aurms_run_fy=aurms_run;
            VDVu_run_fy=VDVu_run;
            aw_fy=aw;
            awrms_run_fy=awrms_run;
            awrms_fy=awrms;
            VDVw_run_fy=VDVw_run;
            VDVw_fy=VDVw;
        elseif a==2 && m==3
            tim_fz = tim;
            amp_fz=amp;
            aurms_run_fz=aurms_run;
            VDVu_run_fz=VDVu_run;
            aw_fz=aw;
            awrms_run_fz=awrms_run;
            awrms_fz=awrms;
            VDVw_run_fz=VDVw_run;
            VDVw_fz=VDVw;
        elseif a==3 && m==1
            tim_px = tim;
            amp_px=amp;
            aurms_run_px=aurms_run;
            VDVu_run_px=VDVu_run;
            aw_px=aw;
            awrms_run_px=awrms_run;
            awrms_px=awrms;
            VDVw_run_px=VDVw_run;
            VDVw_px=VDVw;
        elseif a==3 && m==2
            tim_py = tim;
            amp_py=amp;
            aurms_run_py=aurms_run;
            VDVu_run_py=VDVu_run;
            aw_py=aw;
            awrms_run_py=awrms_run;
            awrms_py=awrms;
            VDVw_run_py=VDVw_run;
            VDVw_py=VDVw;
        elseif a==3 && m==3
            tim_pz = tim;
            amp_pz=amp;
            aurms_run_pz=aurms_run;
            VDVu_run_pz=VDVu_run;
            aw_pz=aw;
            awrms_run_pz=awrms_run;
            awrms_pz=awrms;
```

```
            VDVw_run_pz=VDVw_run;
            VDVw_pz=VDVw;
        end
    end
    %
    if a==1
        VDVw_bxyz=((VDVw_bx^4)+(VDVw_by^4)+(VDVw_bz^4))^(.25)
        VDVw_bxyz_Health=((kxh^4)*(VDVw_bx^4)+(kyh^4)*(VDVw_by^4)+
(kzh^4)*(VDVw_bz^4))^(.25)
        VDVw_bxyz_Comfort=((kxc^4)*(VDVw_bx^4)+(kyc^4)*(VDVw_by^4)+
(kzc^4)*(VDVw_bz^4))^(.25)
        RMSw_bxyz=((kxh^2)*(awrms_bx^2)+(kyh^2)*(awrms_by^2)+(kzh^2)*(awrms_bz^2))^(.
5)
    elseif a==2
        VDVw_fxyz=((VDVw_fx^4)+(VDVw_fy^4)+(VDVw_fz^4))^(.25)
        VDVw_fxyz_Health=((kxh^4)*(VDVw_fx^4)+(kyh^4)*(VDVw_fy^4)+
(kzh^4)*(VDVw_fz^4))^(.25)
        VDVw_fxyz_Comfort=((kxc^4)*(VDVw_fx^4)+(kyc^4)*(VDVw_fy^4)+
(kzc^4)*(VDVw_fz^4))^(.25)
        RMSw_fxyz=((kxh^2)*(awrms_fx^2)+(kyh^2)*(awrms_fy^2)+(kzh^2)*(awrms_fz^2))^(.
5)
    elseif a==3
        VDVw_pxyz=((VDVw_px^4)+(VDVw_py^4)+(VDVw_pz^4))^(.25)
        VDVw_pxyz_Health=((kxh^4)*(VDVw_px^4)+(kyh^4)*(VDVw_py^4)+
(kzh^4)*(VDVw_pz^4))^(.25)
        VDVw_pxyz_Comfort=((kxc^4)*(VDVw_px^4)+(kyc^4)*(VDVw_py^4)+
(kzc^4)*(VDVw_pz^4))^(.25)
        RMSw_pxyz=((kxh^2)*(awrms_px^2)+(kyh^2)*(awrms_py^2)+(kzh^2)*(awrms_pz^2))^(.
5)
    end
end
```

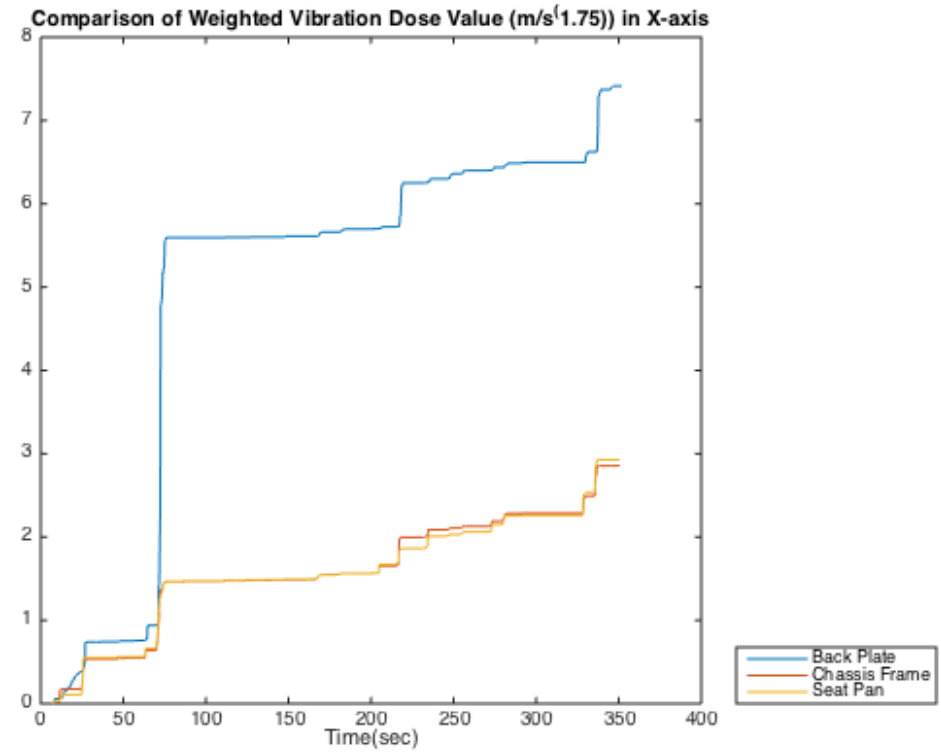
```
  Based on th_weighted.m  ver 1.2  May 31, 2013 by Tom Irvine  Email:
tom@vibrationdata.com
  Adapthed by LNard Tufts  March 24, 2015

  This program converts an acceleration time history to a
  weighted format per ISO 2631.


VDVw_bxyz =

    7.6740


VDVw_bxyz_Health =

    5.9567


VDVw_bxyz_Comfort =

    5.9567
```

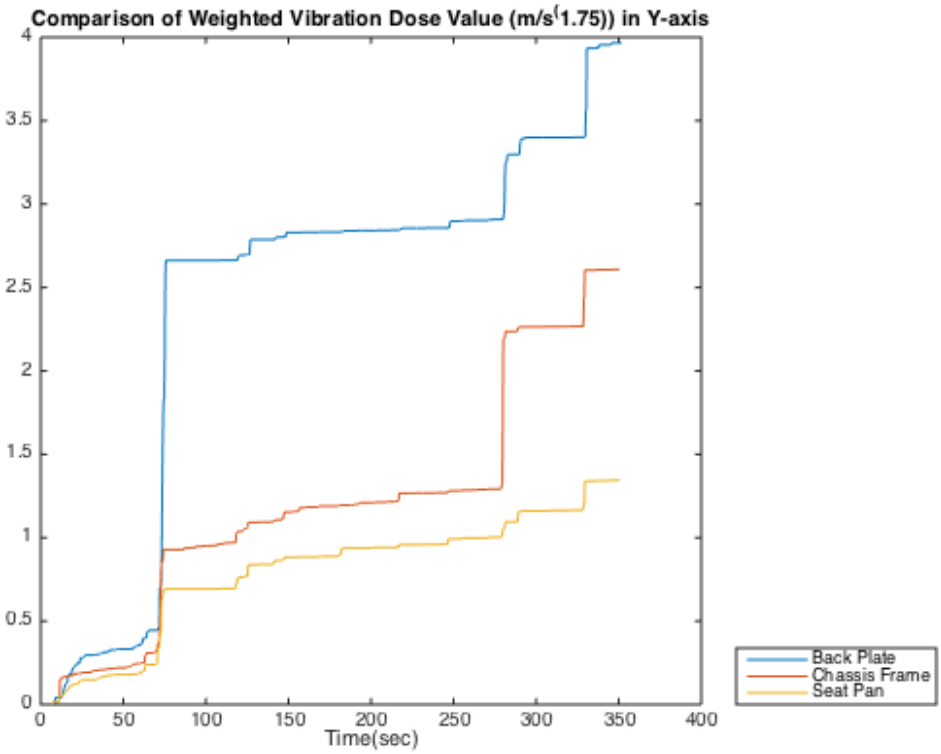
```
RMSw_bxyz =

    0.5736


VDVw_fxyz =

    7.5616


VDVw_fxyz_Health =

    7.7403


VDVw_fxyz_Comfort =

    7.5616


RMSw_fxyz =

    0.6857


VDVw_pxyz =

    4.9405


VDVw_pxyz_Health =

    5.3412


VDVw_pxyz_Comfort =

    4.9405


RMSw_pxyz =

    0.5810
```

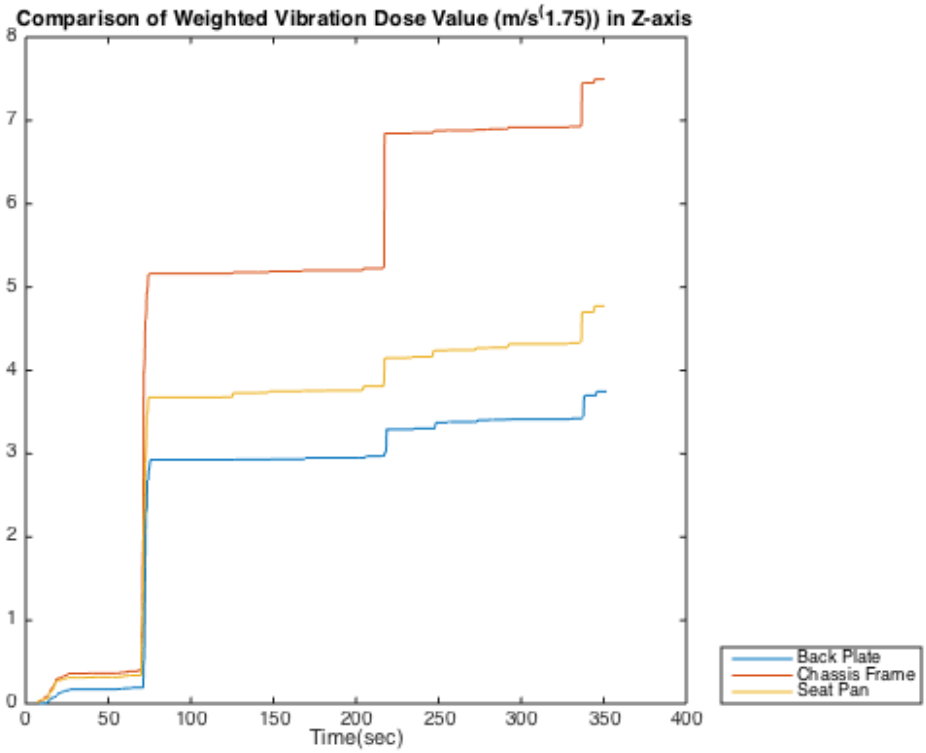
```
figure
plot(tim_bx,VDVw_run_bx,tim_fx,VDVw_run_fx,tim_px,VDVw_run_px)
title('Comparison of Weighted Vibration Dose Value (m/s^(1.75)) in X-axis')
legend('Back Plate','Chassis Frame','Seat Pan','Location','SouthEastOutside')
xlabel('Time(sec)');

out1=sprintf('\n\n Chassis Frame to Seat Pan SEAT Value in X-Axis  SEAT =
%3.1f',VDVw_px*100/VDVw_fx);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support SEAT Value in X-Axis  SEAT =
%3.1f',VDVw_bx*100/VDVw_fx);
disp(out1);
```

```
disp('**********************************************************************************
****************')
%
figure
plot(tim_by,VDVw_run_by,tim_fy,VDVw_run_fy,tim_py,VDVw_run_py)
title('Comparison of Weighted Vibration Dose Value (m/s^(1.75)) in Y-axis')
legend('Back Plate','Chassis Frame','Seat Pan','Location','SouthEastOutside')
xlabel('Time(sec)');

out1=sprintf('\n\n Chassis Frame to Seat Pan SEAT Value in Y-Axis  SEAT =
%3.1f',VDVw_py*100/VDVw_fy);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support SEAT Value in Y-Axis  SEAT =
%3.1f',VDVw_by*100/VDVw_fy);
disp(out1);
disp('**********************************************************************************
****************')
%
figure
plot(tim_bz,VDVw_run_bz,tim_fz,VDVw_run_fz,tim_pz,VDVw_run_pz)
title('Comparison of Weighted Vibration Dose Value (m/s^(1.75)) in Z-axis')
legend('Back Plate','Chassis Frame','Seat Pan','Location','SouthEastOutside')
xlabel('Time(sec)');

out1=sprintf('\n\n Chassis Frame to Seat Pan SEAT Value in Z-Axis  SEAT =
%3.1f',VDVw_pz*100/VDVw_fz);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support SEAT Value in Z-Axis  SEAT =
%3.1f',VDVw_bz*100/VDVw_fz);
disp(out1);
disp('**********************************************************************************
****************')
```

```
 Chassis Frame to Seat Pan SEAT Value in X-Axis  SEAT = 102.5


 Chassis Frame to Back Support SEAT Value in X-Axis  SEAT = 259.7
**********************************************************************************
**********


 Chassis Frame to Seat Pan SEAT Value in Y-Axis  SEAT = 51.5


 Chassis Frame to Back Support SEAT Value in Y-Axis  SEAT = 152.1
**********************************************************************************
**********


 Chassis Frame to Seat Pan SEAT Value in Z-Axis  SEAT = 63.7


 Chassis Frame to Back Support SEAT Value in Z-Axis  SEAT = 50.0
**********************************************************************************
**********
```


Comparison of Weighted Vibration Dose Value (m/s^1.75)) in X-axis
0
1
2
3
4
5
6
7
8
0
50
100
150
200
250
300
350
400
Time(sec)
Back Plate
Chassis Frame
Seat Pan


Comparison of Weighted Vibration Dose Value (m/s^1.75)) in Y-axis
0
0.5
1
1.5
2
2.5
3
3.5
4
0
50
100
150
200
250
300
350
400
Time(sec)
Back Plate
Chassis Frame
Seat Pan




```
out1=sprintf('\n\n Chassis Frame to Seat Pan Overall SEAT Value  SEAT =
%3.1f',VDVw_pxyz*100/VDVw_fxyz);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support Overall SEAT Value  SEAT =
%3.1f',VDVw_bxyz*100/VDVw_fxyz);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Seat Pan Overall SEAT Value (Health Weighted)
SEAT = %3.1f',VDVw_pxyz_Health*100/VDVw_fxyz_Health);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support Overall SEAT Value (Health Weighted)
SEAT = %3.1f',VDVw_bxyz_Health*100/VDVw_fxyz_Health);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Seat Pan Overall SEAT Value (Comfort Weighted)
SEAT = %3.1f',VDVw_pxyz_Comfort*100/VDVw_fxyz_Comfort);
disp(out1);
out1=sprintf('\n\n Chassis Frame to Back Support Overall SEAT Value (Comfort Weighted)
SEAT = %3.1f',VDVw_bxyz_Comfort*100/VDVw_fxyz_Comfort);
disp(out1);
```

```
 Chassis Frame to Seat Pan Overall SEAT Value  SEAT = 65.3


 Chassis Frame to Back Support Overall SEAT Value  SEAT = 101.5


 Chassis Frame to Seat Pan Overall SEAT Value (Health Weighted)  SEAT = 69.0


 Chassis Frame to Back Support Overall SEAT Value (Health Weighted)  SEAT = 77.0
```

```
 Chassis Frame to Seat Pan Overall SEAT Value (Comfort Weighted)  SEAT = 65.3

 Chassis Frame to Back Support Overall SEAT Value (Comfort Weighted)  SEAT = 78.8
```

Published with MATLAB® R2014b

APPENDIX P

Principle and Additional Frequency Weightings[14]

Table 3 — Principal frequency weightings in one-third octaves

Frequency band number[1] x	Frequency f Hz	W_k factor ×1 000	W_k dB	W_d factor ×1 000	W_d dB	W_f factor ×1 000	W_f dB
−17	0,02					24,2	−32,33
−16	0,025					37,7	−28,48
−15	0,031 5					59,7	−24,47
−14	0,04					97,1	−20,25
−13	0,05					157	−16,10
−12	0,063					267	−11,49
−11	0,08					461	−6,73
−10	0,1	31,2	−30,11	62,4	−24,09	695	−3,16
−9	0,125	48,6	−26,26	97,3	−20,24	895	−0,96
−8	0,16	79,0	−22,05	158	−16,01	1 006	0,05
−7	0,2	121	−18,33	243	−12,28	992	−0,07
−6	0,25	182	−14,81	365	−8,75	854	−1,37
−5	0,315	263	−11,60	530	−5,52	619	−4,17
−4	0,4	352	−9,07	713	−2,94	384	−8,31
−3	0,5	418	−7,57	853	−1,38	224	−13,00
−2	0,63	459	−6,77	944	−0,50	116	−18,69
−1	0,8	477	−6,43	992	−0,07	53,0	−25,51
0	1	482	−6,33	1 011	0,10	23,5	−32,57
1	1,25	484	−6,29	1 008	0,07	9,98	−40,02
2	1,6	494	−6,12	968	−0,28	3,77	−48,47
3	2	531	−5,49	890	−1,01	1,55	−56,19
4	2,5	631	−4,01	776	−2,20	0,64	−63,93
5	3,15	804	−1,90	642	−3,85	0,25	−71,96
6	4	967	−0,29	512	−5,82	0,097	−80,26
7	5	1 039	0,33	409	−7,76		
8	6,3	1 054	0,46	323	−9,81		
9	8	1 036	0,31	253	−11,93		
10	10	988	−0,10	212	−13,91		
11	12,5	902	−0,89	161	−15,87		
12	16	768	−2,28	125	−18,03		
13	20	636	−3,93	100	−19,99		
14	25	513	−5,80	80,0	−21,94		
15	31,5	405	−7,86	63,2	−23,98		
16	40	314	−10,05	49,4	−26,13		
17	50	246	−12,19	38,8	−28,22		
18	63	186	−14,61	29,5	−30,60		
19	80	132	−17,56	21,1	−33,53		
20	100	88,7	−21,04	14,1	−36,99		
21	125	54,0	−25,35	8,63	−41,28		
22	160	28,5	−30,91	4,55	−46,84		
23	200	15,2	−36,38	2,43	−52,30		
24	250	7,90	−42,04	1,26	−57,97		
25	315	3,98	−48,00	0,64	−63,92		
26	400	1,95	−54,20	0,31	−70,12		

1) Index x is the frequency band number according to IEC 1260.

NOTES

1 For tolerances of the frequency weightings, see 6.4.1.2.

2 If it has been established that the frequency range below 1 Hz is unimportant to the weighted acceleration value, a frequency range 1 Hz to 80 Hz is recommended.

3 The values have been calculated including frequency band limitation.

Table 4 — Additional frequency weightings in one-third octaves

Frequency band number[1] x	Frequency f Hz	W_c		W_e		W_j	
		factor × 1 000	dB	factor × 1 000	dB	factor × 1 000	dB
− 10	0,1	62,4	− 24,11	62,5	− 24,08	31,0	− 30,18
− 9	0,125	97,2	− 20,25	97,5	− 20,22	48,3	− 26,32
− 8	0,16	158	− 16,03	159	− 15,98	78,5	− 22,11
− 7	0,2	243	− 12,30	245	− 12,23	120	− 18,38
− 6	0,25	364	− 8,78	368	− 8,67	181	− 14,86
− 5	0,315	527	− 5,56	536	− 5,41	262	− 11,65
− 4	0,4	708	− 3,01	723	− 2,81	351	− 9,10
− 3	0,5	843	− 1,48	862	− 1,29	417	− 7,60
− 2	0,63	929	− 0,64	939	− 0,55	458	− 6,78
− 1	0,8	972	− 0,24	941	− 0,53	478	− 6,42
0	1	991	− 0,08	880	− 1,11	484	− 6,30
1	1,25	1 000	0,00	772	− 2,25	485	− 6,28
2	1,6	1 007	0,06	632	− 3,99	483	− 6,32
3	2	1 012	0,10	512	− 5,82	482	− 6,34
4	2,5	1 017	0,15	409	− 7,77	489	− 6,22
5	3,15	1 022	0,19	323	− 9,81	524	− 5,62
6	4	1 024	0,20	253	− 11,93	628	− 4,04
7	5	1 013	0,11	202	− 13,91	793	− 2,01
8	6,3	974	− 0,23	160	− 15,94	946	− 0,48
9	8	891	− 1,00	125	− 18,03	1 017	0,15
10	10	776	− 2,20	100	− 19,98	1 030	0,26
11	12,5	647	− 3,79	80,1	− 21,93	1 026	0,22
12	16	512	− 5,82	62,5	− 24,08	1 018	0,16
13	20	409	− 7,77	50,0	− 26,02	1 012	0,10
14	25	325	− 9,76	39,9	− 27,97	1 007	0,06
15	31,5	256	− 11,84	31,6	− 30,01	1 001	0,00
16	40	199	− 14,02	24,7	− 32,15	991	− 0,08
17	50	156	− 16,13	19,4	− 34,24	972	− 0,24
18	63	118	− 18,53	14,8	− 36,62	931	− 0,62
19	80	84,4	− 21,47	10,5	− 39,55	843	− 1,48
20	100	56,7	− 24,94	7,07	− 43,01	708	− 3,01
21	125	34,5	− 29,24	4,31	− 47,31	539	− 5,36
22	160	18,2	− 34,80	2,27	− 52,86	364	− 8,78
23	200	9,71	− 40,26	1,21	− 58,33	243	− 12,30
24	250	5,06	− 45,92	0,63	− 63,99	158	− 16,03
25	315	2,55	− 51,88	0,32	− 69,94	100	− 19,98
26	400	1,25	− 58,08	0,16	− 76,14	62,4	− 24,10

1) Index x is the frequency band number according to IEC 1260.

NOTES

1 For tolerances of the frequency weightings, see 6.4.1.2.

2 If it has been established that the frequency range below 1 Hz is unimportant to the weighted acceleration value, a frequency range 1 Hz to 80 Hz is recommended.

3 The values have been calculated including frequency band limitation.

References

1. Bremseth law firm; Available from: http://www.bremseth.com/Railroad-Injuries/Whole-Body- Vibration-Injuries.shtml. Accessed 3 February 2015.
2. Mansfield NJ. Human response to vibration. Boca Raton, FL: CRC Press; 2005.
3. Pearlman J. What we know and need to find out about the health implications of vibrations on wheelchair users. Salon 2010:62-5.
4. Boninger ML, Cooper RA, Fitzgerald SG, Lin J, Cooper R, Dicianno B, Liu B. Investigating neck pain in wheelchair users. Am J Phys Med Rehabil 2003, Mar;82(3):197-202.
5. Evaluation of electric powered wheelchairs and exposure to whole-body vibration - introduction; Proceedings of the first american conference on human vibration. 2006.
6. Disabled world; Available from: http://www.disabled-world.com/disability/statistics/american-disability.php. Accessed 2 February 2015.
7. Sonenblum SE, Sprigle S, Maurer C. Monitoring power upright and tilt-in-space wheelchair use. 2006.
8. Available from: http://www.arduino.cc. Accessed 21 February 2015.
9. XBee buying guide. : Available from: https://www.sparkfun.com/pages/xbee_guide.
10. Jimb0 J. Available from: https://learn.sparkfun.com/tutorials/arduino-shields. Accessed 23 February 2015.
11. Faludi R. Building wireless sensor networks: With zigbee, xbee, arduino, and processing. O'Reilly Media, Inc.; 2010.
12. Earl B. Adafruit analog accelerometer breakouts. : Available from: https://learn.adafruit.com/adafruit-analog-accelerometer-breakouts?view=all.
13. Company EK. Kodak's ergonomic design for people at work. Hoboken, NJ: Wiley; 2004.
14. ISO. Mechanical vibration and shock: Evaluation of human exposure to whole-body vibration. Part 1, general requirements: International standard ISO 2631-1: 1997 (E). ISO; 1997.
15. DiGiovine CP, Cooper RA, Fitzgerald SG, Boninger ML, Wolf EJ, Guo S. Whole-body vibration during manual wheelchair propulsion with selected seat cushions and back supports. IEEE Trans Neural Syst Rehabil Eng 2003, Sep;11(3):311-22.
16. Garcia-Mendez Y, Pearlman JL, Cooper RA, Boninger ML. Dynamic stiffness and transmissibility of commercially available wheelchair cushions using a laboratory test method. J Rehabil Res Dev 2012;49(1):7-22.
17. Arduino_uno_R3_pinout.Jpg. : Available from: http://thietbichetao.com/wp-content/uploads/2014/03/arduino_uno_R3_pinout.jpg.